Statistical Inferences II (Confidence Interval)

Lucie Lu

May 13, 2018

Confidence Invertals

In the previous section, I have obtained the p-value by repeatedly calculating the test statistic from a large number of permutations of the data. On its surface, the permutation confidence interval in this section is simply the set of all values of the parameter for which the null hypothesis is not rejected. To obtain a confidence interval, I tried to use some p-values to test different hypotheses to create a collection of non-rejected hypotheses. This is the confidence interval from the permutation tests.

My null hypothesis are addictive treatment effects. I used two methods to calculate the confidence intervals: one with the uniroot function to find two tails of confidence limits where the p value from the *lm* or *lmrob* function of certain hypothesis is less than 5%, leading to the rejection of that hypothesis. This one is a CLT+IID based test.

The second test sets the significance level 0.05, and specify the confidence coefficient 0.95 to reflect the true coverage probability. I created null hypotheses as a constant addictive effect and compute an intervavl estimate where the right and left end points' p values are approximately equal to 0.05. Within the two-sided confidence intervals, the estimates' p-values are larger than 0.05. I show part of the procedure in this method in table "Searching for non-rejected intervals under permuation for lm and lmrob models."

From the table we can see this second method (see the results in the "permuted lm confidence interval" and "permuted lmrob confidence interval") produces confidence intervals that do not accord with the first method. This may be due to some coding issues that I cannot figure it out at this moment.

	1	2	3	4	5	6	7	8	9	10	11	12	13
additive treatment effect	-10.00	-9.00	-8.00	-7.00	-6.00	-5.00	-4.00	-3.00	-2.00	-1.00	0.00	1.00	2.00
p-values for lm	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.04	0.11	0.28	0.45	0.73
p-values for lmrob	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.02	0.06	0.17	0.50

	2.5 %	97.5 %
standard lm confidence interval	-8.07	2.40
uniroot lm confidence interval	-8.07	2.40
permuted lm confidence interval	-1.00	7.00
standard lmrob confidence interval	-6.50	-0.12
uniroot lmrob confidence interval	-6.50	-0.12
permuted lmrob confidence interval	0.00	6.00

	Variance–covariance matrix	Sandwich package
Standard OLS	2.68	2.68
HC0	2.53	2.53
HC1	2.57	2.57
HC2	2.58	2.58
HC3	2.62	2.62

	Robust Standard Errors
Clustered HC0	2.53
Clustered HC1	2.57
Clustered HC2	2.58
Clustered HC3	2.62
Clustered SEs from Variance-Covariance matrix	2.57
Wild Bootstrap	2.64
Clustered Wild Bootstrap	2.49

The standard errors drawn from the OLS canned function will be biased if there is a presence of heteroskedasticity of unknown form in my dataset. Note that the assumption the variance of the error term for each x is constant (Homoskedasticity) is not necessary to show that OLS estimators are unbiased. Heteroscedasticity occurs when the variance of the errors varies across observations (Long & Ervin, 2000, p.217). In simple words, when the variance of errors is not constant, one of the important assumptions of a linear model, homoscedasticity is violated. In the presence of heteroscedasticity, ordinary least squares estimates are still consistent, but the tests of significance are generally inappropriate because the standard errors obtained from the classic variance covariance matrix are no longer consistent. We will then perform an invalid statistical inference if we do not correct for the possible presence of heteroskedasticity. Figure 2 reveals that we might have a heteroskedasticity problem.

Here, I use a avariety of approaches to correct OLS standard errors, by hands and by using packages. The table shows that the results are similar.

I simplify my regression without controlling for covariates as the following at this moment: $tradepolicy_{i,t} = \beta_0 + \beta_1 * DEMOCRACY_{i,t-1} + \varepsilon_i$.

I calculated the unclusterd HC0, HC1, HC2, and HC3 robust standard errors by heteroskedasticity consistent covariance matrix (HCCME) (also known as White robust errors) and by using a "sandwich" R-statistic package. The results are the same, so I know the results from the package are not misleading. I also use the "wild bootstrap" simulation method to cross-check the results. The "HC" standard errors taking into account the Heteroscedasticity and the by-default standard errors are in fact very close.

The clustered versions of standard errors from different methods are also very close to the unclustered versions.

```r
## To make the pdf file do
## render("exploration4.Rmd",output_format=pdf_document())

require(knitr)
opts_chunk$set(
  tidy=FALSE,     # display code as typed
  size="small",    # slightly smaller font for code
  echo=TRUE,
  results='markup',
  strip.white=TRUE,
  fig.path='figs/fig',
  cache=FALSE,
  highlight=TRUE,
  width.cutoff=132,
  size='footnotesize',
  out.width='.9\\textwidth',
  message=FALSE,
  comment=NA)

##First, just setup the R environment for today:
if(!file.exists('figs')) dir.create('figs')

options(SweaveHooks=list(fig=function(){
```

```r
               par(mar=c(3.5, 3, 1.1, 0),
                   pty="s",
                   mgp=c(1.5,0.5,0),
                   oma=c(0,0,0,0))},
             echo=function(){options(continue=" ") ##Don't show "+" prompts,
             options(prompt=" ")
             }),
    digits=4,
    scipen=8,
    width=132
    )
options(error=function(){options(prompt="> ",continue="+ ");NULL})


#To prepare the environment
library(readstata13)
library(dplyr)
library(ggplot2)
#install.packages("ggrepel")
library(ggrepel)
library(stargazer)

#install.packages("devtools")
library(devtools)

library(foreign)
#install.packages("gplots")
library(gplots)

library(lmtest)
library(sandwich)

library(tidyverse)

library(MASS)
library(robustbase)

#install.packages("rmngb")
library(rmngb)
library(here)

#devtools::install_github("ropenscilabs/gramr")
library("gramr")
#write_good_ip()

library(xtable)

getwd()
set_here(path=".", verbose=T)
here()

list.files(path=".")
load("mydata_5collapseyear.rda")
```

```r
OLS_5_0Bs <- lm(Tariff_c ~ Regime_Bi_c, data=mydata_5)
summary(OLS_5_0Bs)$coefficient[2,1] #-4.29

OLS_5_5Bs <- lm(Tariff_c ~ Regime_Bi_c + GDP_c + EC_c + WTO_c, data=mydata_5)
summary(OLS_5_5Bs)$coefficient[2,1] #-2.84

############


#z is the experiment, shuffle the mydata4$Regime_l1_Bi variable to create a pretend experiment

set.seed(2018)

mydata_5test <- mydata_5

#rm(mydata_5test$a)
#mydata_5test %>% select(-a)
mydata_5test$Z <- sample(rep(c(0,1), each = 1, len = 81))



#'A confidence interval is a collection of not-rejected hypotheses.'
#Under lm:
OLS_5_5Bs <- lm(Tariff_c ~ Regime_Bi_c + GDP_c + EC_c + WTO_c, data=mydata_5)

#Under lm assumption:
confint(OLS_5_5Bs, parm="Regime_Bi_c")
cicanned_lm <- round(confint(OLS_5_5Bs, parm="Regime_Bi_c"),2)


#############
mytest2ForUniRoot_test<-function(x,y=mydata_5$Tariff_c,z=mydata_5$Regime_Bi_c){
             newy<-y-(z*x)
             .05-summary(lm(newy~ z + GDP_c + EC_c + WTO_c, data=mydata_5))$coef["z","Pr(>|t|)"]
}

#mytest2ForUniRoot_test(20)
#mytest2ForUniRoot(-20)
upperlim<-uniroot(mytest2ForUniRoot_test,interval=c(0,20),extendInt="no")
lowerlim<-uniroot(mytest2ForUniRoot_test,interval=c(-20,0),extendInt="no")
ciroot_lm <- round(c(lowerlim$root,upperlim$root),2)


#This is another way to find the confidence interval using uniroot. The function here is still about ea


### The root means the solution that makes a function equal to 0, and uniroot command is telling R to fi
###"extendInt=No" is telling R not extend search if a different sign cannot be found. In other words, re

cicanned_lm
ciroot_lm

#'A confidence interval is a collection of not-rejected hypotheses.'
```

```
#Under lmrob:
lmrob_5Bs <- lmrob(Tariff_c ~ Regime_Bi_c + GDP_c + EC_c + WTO_c, data=mydata_5)

confint(lmrob_5Bs, parm="Regime_Bi_c")

cicanned_lmrob <- round (confint(lmrob_5Bs, parm="Regime_Bi_c"), 2)

##############

mytest3ForUniRoot_test<-function(x,y=mydata_5$Tariff_c,z=mydata_5$Regime_Bi_c){
             newy<-y-(z*x)
             .05-summary(lmrob(newy~ z + GDP_c + EC_c + WTO_c, data=mydata_5))$coef["z","Pr(>|t|)"]
}

#mytest3ForUniRoot_test(20)
#mytest3ForUniRoot_test(-20)
###First, the root means the solution that makes
#a function equal to 0, and uniroot command is telling R
#to find only one solution. Second, the upperlim and lowerlim
#represent the two-tail test. Both upperlim and lowerlim
#tell R to search only one root from the interval 0 to 20 or -20 to 0.
### "extendInt=No" is telling R not extend search if a different sign cannot be found. In other words, i


upperlim<-uniroot(mytest3ForUniRoot_test,interval=c(0,20),extendInt="yes")
lowerlim<-uniroot(mytest3ForUniRoot_test,interval=c(-20,-0.9),extendInt="yes")
ciroot_lmrob <- round(c(lowerlim$root,upperlim$root),2)

cicanned_lmrob
ciroot_lmrob

## Confidence interval: choose alpha=.05
set.seed(2349854)

OLS_5_5Bs <- lm(Tariff_c ~ Regime_Bi_c + GDP_c + EC_c + WTO_c, data=mydata_5)
simlmtest_lm5Bs <- summary(OLS_5_5Bs)$coef[2,1]
simlmtest_lm5Bs #-2.386


#Imagine we re-run the "experiment"
newExp_lm<-function(y){
  ## A function to randomly assign treatment effect
shuffledz <- sample(mydata_5test$Z)
newlmtest <- coef(lm(y ~ shuffledz+ GDP_c + EC_c + WTO_c, data=mydata_5test))[["shuffledz"]]
  return(newlmtest)
}

myTestStat4_lm<-function(x,newz=shuffledz,y=mydata_5test$Tariff_c){
  shuffledz <- sample(mydata_5test$Z)
  newy<-y-(newz*x)
    coef(lm(newy~ newz + GDP_c + EC_c + WTO_c, data=mydata_5test))[["newz"]]
}
```

```r
MyFisherTest4_lm <- function(x,thez){
  ## return a p-value
  randDistH0<-replicate(1000,myTestStat4_lm(x=x))
  pTwoSided <- 2*min(c(mean(randDistH0>=simlmtest_lm5Bs),
                mean(randDistH0<=simlmtest_lm5Bs)))
  return(pTwoSided)
}
#######
######

library(foreach)
res1<-foreach(h=seq(-10, 10, 1),.combine='c') %dopar% {message("."); MyFisherTest4_lm(x=h, thez=mydata_
#Now I can use foreach to execute the function repeatedly, passing it the values -10 through 5, and ret


printCIres1lm <- rbind(seq(-10, 10, 1),res1)
printCIres1lm

#CI for lm by using using permutation here is [-1, 7].

lmrob_5Bs <- lmrob(Tariff_c ~ Regime_Bi_c + GDP_c + EC_c + WTO_c, data=mydata_5, control = lmrob.contro

simlmrobtest_5Bs <- summary(lmrob_5Bs)$coef[2,1]
simlmrobtest_5Bs #-3.31

## Confidence interval: choose alpha=.05
set.seed(2349854)
#Imagine we re-run the "experiment"

newExp_lmrob<-function(y){
  ## A function to randomly assign treatment effect
shuffledz <- sample(mydata_5test$Z)
newlmtest <- coef(lmrob(y ~ shuffledz + GDP_c + EC_c + WTO_c, data=mydata_5test))[["shuffledz"]]
  return(newlmtest)
}

myTestStat4_lmrob<-function(x,newz=shuffledz,y=mydata_5test$Tariff_c){
  shuffledz <- sample(mydata_5test$Z)
  newy<-y-(newz*x)
    coef(lmrob(newy ~ newz + GDP_c + EC_c + WTO_c, data=mydata_5test))[["newz"]]
    #return(coef)
}


MyFisherTest4_lmrob <- function(x,thez){
  ## return a p-value
  randDistH0<-replicate(1000,myTestStat4_lmrob(x=0))
  pTwoSided <- 2*min(c(mean(randDistH0>=simlmrobtest_5Bs),
                mean(randDistH0<=simlmrobtest_5Bs)))
  return(pTwoSided)
}
```

```
MyFisherTest4_lmrob <- function(x,thez){
  ## return a p-value
  randDistH0<-replicate(1000,myTestStat4_lmrob(x=x))
  pTwoSided <- 2*min(c(mean(randDistH0>=simlmrobtest_5Bs),
              mean(randDistH0<=simlmrobtest_5Bs)))
  return(pTwoSided)
}
#######
######
library(foreach)
res1_lmrob<-foreach(h=seq(-10,10,1),.combine='c') %dopar% {message("."); MyFisherTest4_lmrob(x=h, thez=


printCIres1lmrob <- rbind(seq(-10,10,1),res1_lmrob)
printCIres1lmrob

#CI here is [0,3].

ci_permute <- round(as.matrix(rbind(printCIres1lm, res1_lmrob)),2)
#ci_permute
rownames(ci_permute) <- c("additive treatment effect", "p-values for lm", "p-values for lmrob")
xtable(ci_permute)


print(xtable::xtable(ci_permute, caption = "Searching for non-rejected intervals under permuation for l
      html.table.attributes="border=1", comment=F, width = "\\textwidth", floating = F)
ci_permute_lm <- c(-1, 7)
ci_permute_lmrob <- c(0, 6)

collect_ci_test <- as.matrix(rbind(cicanned_lm, ciroot_lm, ci_permute_lm, cicanned_lmrob,ciroot_lmrob, 

rownames(collect_ci_test) <- c("standard lm confidence interval", "uniroot lm confidence interval", "pe

collect_ci_test


print(xtable::xtable(collect_ci_test, caption = "Collections of Confudence Intervals under pumutation f
      html.table.attributes="border=1", comment=F, floating = F)

#OLS_5_5Bs <- lm(Tariff_c ~ Regime_Bi_c + GDP_c + EC_c + WTO_c, data=mydata_5)

OLS_5_0Bs <- lm(Tariff_c ~ Regime_Bi_c, data=mydata_5)
summary(OLS_5_0Bs)

#X <- model.matrix(~Regime_l1, data=mydata4_small)#use the discrete polity score
#nrow(X) #3246 observations

X_m5 <- model.matrix(~Regime_Bi_c, data=mydata_5test)#make a model matrix
dim(X_m5) #81

y_m5<- mydata_5test$Tariff_c
N <- length(y_m5) #81
```

```
b_m5<- solve(t(X_m5) %*% X_m5) %*% t(X_m5) %*% y_m5 #coefficient Regime: NA because there are missing v
b_m5 #-4.287

ehat_m5<-y_m5 - (X_m5 %*% b_m5)
names(ehat_m5)<-row.names(mydata_5test)
sigma2_m5<-sum(ehat_m5^2)/(nrow(mydata_5test)-length(b_m5))
#sum(ehat^2) is the sum of residuals sqaures; sigam^2 is the variance of the random errors, which equal

vcovb_m5<- sigma2_m5 * solve(t(X_m5) %*% X_m5)
vcovb_m5
#This is covariance matrix for the estimated coefficients.

seb_m5<-sqrt(diag(vcovb_m5))
seb_m5
#This is the standard errors for estimated coefficients.

cbind(b_m5,seb_m5)
#This is the matrix of coefficients and standard errors


#Performs the (formal) Breusch-Pagan test against heteroskedasticity.
bptest(Tariff_c ~ Regime_Bi_c, data=mydata_5)
#The Breusch-Pagan test fits a linear regression model to the residuals of a linear regression model.
#The residuals should not be well explained by the predictors. My null hypothesis here is variance of e
#The p-value is 0.8, which means that we do not reject against the null hypothesis of homoscedasticity.
OLS_5_OBs <- lm(Tariff_c ~ Regime_Bi_c, data=mydata_5)
summary(OLS_5_OBs)

#We use Long & Ervin (2000) and the code, along with the sandwich package to calculate HC0... etc.
H_m5 <- diag(X_m5 %*% solve(t(X_m5) %*% X_m5) %*% t(X_m5)) #This is leverage, Leverage tells us how muc

#install.packages("sandwich")
library(sandwich)

##OLS standard Errors
## We need an N x N matrix in the middle of this next calculation
vcovIID_m5 <- (solve(crossprod(X_m5)) %*% t(X_m5)) %*% (sigma2_m5 * diag(1,N,N))%*% (X_m5%*%solve(cross
#vcov(OLS_4_0) by default function
sebIID_m5  <- sqrt(diag(vcovIID_m5))

sebIID_m5 #this is the same as the coeff from the canned regression under the assumption of homoscedast

sandseb_m5<- sqrt(diag(vcovHC(OLS_5_OBs, type = "const")))
sandseb_m5 #sebIID_m5 == sandseb_m5


## Make a block diagonal sigma/middle matrix
thesigmaHC0_m5<-diag(as.vector(ehat_m5)^2,nrow=length(ehat_m5),ncol=length(ehat_m5))
dimnames(thesigmaHC0_m5)<-list(names(ehat_m5),names(ehat_m5))

##HC0
vcovHC0_m5 <- (solve(crossprod(X_m5)) %*% t(X_m5)) %*% (thesigmaHC0_m5) %*% ( X_m5 %*% solve(crossprod(
```

```r
sebHC0_m5<-sqrt(diag(vcovHC0_m5))
sandsebHC0_m5 <- sqrt(diag(vcovHC(OLS_5_OBs, type = "HC0")))

sebHC0_m5
sandsebHC0_m5

##HC1
vcovHC1_m5 <- (N/(N-2))*vcovHC0_m5 ## 2 coeff: beta1, intercept

sebHC1_m5 <-sqrt(diag(vcovHC1_m5))
sandsebHC1_m5 <- sqrt(diag(vcovHC(OLS_5_OBs, type = "HC1")))

sebHC1_m5
sandsebHC1_m5

##HC2
thesigmaHC2_m5<-diag(as.vector(ehat_m5)^2/(1-H_m5),nrow=length(ehat_m5),ncol=length(ehat_m5))
dimnames(thesigmaHC2_m5)<-list(names(ehat_m5),names(ehat_m5))

vcovHC2_m5 <- (solve(crossprod(X_m5)) %*% t(X_m5)) %*% (thesigmaHC2_m5) %*% ( X_m5 %*% solve(crossprod(
##car::hccm(OLS_4_0,type="hc2")

sebHC2_m5 <- sqrt(diag(vcovHC2_m5))
sandsebHC2_m5 <- sqrt(diag(vcovHC(OLS_5_OBs, type = "HC2")))

sebHC2_m5
sandsebHC2_m5

#HC3 Long & Ervin prefer HC3 write that HC3 divides ehat^2 by (1-h)^2 further inflates ehat^2 which adj
thesigmaHC3_m5<-diag(as.vector(ehat_m5)^2/(1-H_m5)^2,nrow=length(ehat_m5),ncol=length(ehat_m5))
dimnames(thesigmaHC3_m5)<-list(names(ehat_m5),names(ehat_m5))

vcovHC3_m5 <- (solve(crossprod(X_m5)) %*% t(X_m5)) %*% (thesigmaHC3_m5) %*% ( X_m5 %*% solve(crossprod(

#car::hccm(OLS_4_0,type="hc3")
#These are also called White-corrected or White-Huber covariance matrices.

sebHC3_m5 <- sqrt(diag(vcovHC3_m5))
sandsebHC3_m5 <- sqrt(diag(vcovHC(OLS_5_OBs, type = "HC3")))

sebHC3_m5
sandsebHC3_m5

#options(digits=4)
#Without considering clustering
SEOLS_m5 <- c(sandseb_m5[2], summary(OLS_5_OBs)$coef[2,2])
SEhc0_m5 <- c(sebHC0_m5[2], sandsebHC0_m5[2])
SEhc1_m5 <- c(sebHC1_m5[2], sandsebHC1_m5[2])
SEhc2_m5 <- c(sebHC2_m5[2], sandsebHC2_m5[2])
SEhc3_m5 <- c(sebHC3_m5[2], sandsebHC3_m5[2])

SEcol1<- rbind(SEOLS_m5, SEhc0_m5, SEhc1_m5, SEhc2_m5, SEhc3_m5)
SEcol1
```

```r
cname_SEcol1 <- c("Variance-covariance matrix", "Sandwich package")
rname_sEcol1 <- c("Standard OLS", "HC0", "HC1", "HC2", "HC3")
rownames(SEcol1) <- rname_sEcol1
colnames(SEcol1) <- cname_SEcol1

SEcol1
xtable(SEcol1)
print(xtable::xtable(SEcol1, caption = "Calculating Standard Errors under Heteroskedasticity"),type = "
      html.table.attributes="border=1", comment=F, floating = F)


library(multiwayvcov)

#Clustering in consideration
## For comparison, produce White HC0 VCOV the hard way
vcov_hc0_cm5 <- cluster.vcov(OLS_5_OBs, 1:nrow(mydata_5test), df_correction = FALSE)
coeftest(OLS_5_OBs, vcov_hc0_cm5)
## Produce White HC1 VCOV the hard way
vcov_hc1_cm5 <- cluster.vcov(OLS_5_OBs, 1:nrow(mydata_5test), df_correction = TRUE)
coeftest(OLS_5_OBs, vcov_hc1_cm5)
## Produce White HC2 VCOV the hard way
vcov_hc2_cm5 <- cluster.vcov(OLS_5_OBs, 1:nrow(mydata_5test), df_correction = FALSE, leverage = 2)
coeftest(OLS_5_OBs, vcov_hc2_cm5)
## Produce White HC3 VCOV the hard way
vcov_hc3_cm5 <- cluster.vcov(OLS_5_OBs, 1:nrow(mydata_5test), df_correction = FALSE, leverage = 3)
coeftest(OLS_5_OBs, vcov_hc3_cm5)


mydata_5test$CountryF <- as.factor(mydata_5test$Country)

##Comments: there are not many differences between clustered and non-clustered HC0, HC1, HC2, and HC3.

##Consider clustering with cluster.cov
# Cluster by country
vcov_cm5_c <- cluster.vcov(OLS_5_OBs, mydata_5test$CountryF)
coeftest(OLS_5_OBs, vcov_cm5_c)


######try cluster functions

robustse_test <- function(model, cluster){
 require(sandwich)
 require(lmtest)
 M <- length(unique(cluster))
 N <- length(cluster)
 K <- model$rank
 dfc <- (M/(M - 1)) * ((N - 1)/(N - K))
 uj <- apply(estfun(model), 2, function(x) tapply(x, cluster, sum));
 rcse.cov <- dfc * sandwich(model, meat = crossprod(uj)/N)
 rcse.se <- coeftest(model, rcse.cov)
 return(list(rcse.cov, rcse.se))
}
```

```r
vcov <- robustse_test(OLS_5_0Bs, mydata_5test$CountryF)[[1]]
coefs <- robustse_test(OLS_5_0Bs, mydata_5test$CountryF)[[2]]

OLS_5_5Bs <- lm(Tariff_c ~ Regime_Bi_c + GDP_c + EC_c + WTO_c, data=mydata_5)
summary(OLS_5_5Bs)

robustse_test <- function(model, cluster){
 require(sandwich)
 require(lmtest)
 M <- length(unique(cluster))
 N <- length(cluster)
 K <- model$rank
 dfc <- (M/(M - 1)) * ((N - 1)/(N - K))
 uj <- apply(estfun(model), 2, function(x) tapply(x, cluster, sum));
 rcse.cov <- dfc * sandwich(model, meat = crossprod(uj)/N)
 rcse.se <- coeftest(model, rcse.cov)
 return(list(rcse.cov, rcse.se))
}

vcov <- robustse_test(OLS_5_5Bs, mydata_5test$CountryF)[[1]]
coefs <- robustse_test(OLS_5_5Bs, mydata_5test$CountryF)[[2]]

vcov
coefs

#summary(OLS_5_5Bs)
# Wild Bootstrap.
#https://cran.r-project.org/web/packages/multiwayvcov/multiwayvcov.pdf


set.seed(19900814)
#install.packages("multiwayvcov")
library(multiwayvcov)
library(lmtest)

OLS_5_0Bs <- lm(Tariff_c ~ Regime_Bi_c, data=mydata_5)
summary(OLS_5_0Bs)

# Wild Bootstrap
vcovWB_m5 <- cluster.boot(OLS_5_0Bs, cluster=1:nrow(mydata_5test),boot_type="wild")
coeftest(OLS_5_0Bs,vcovWB_m5)

# Clustered Wild Bootstrap
##Cluster by country
vcovWBC_m5_c <- cluster.boot(OLS_5_0Bs,cluster=mydata_5test$CountryF,boot_type="wild")
coeftest(OLS_5_0Bs,vcovWBC_m5_c) # Wild Bootstrap with Clustering by country

#When considering clustering


#robust covariance matrix vcovHC
SE_c_hc0 <- sqrt((vcov_hc0_cm5)[2,2])
SE_c_hc1 <- sqrt((vcov_hc1_cm5)[2,2])
```

```
SE_c_hc2 <- sqrt((vcov_hc2_cm5)[2,2])
SE_c_hc3 <- sqrt((vcov_hc3_cm5)[2,2])
SE_c_func <- robustse_test(OLS_5_OBs, mydata_5test$CountryF)[[2]][2,2]
SE_w <- sqrt((vcovWB_m5)[2,2])
SE_c_w <- sqrt((vcovWBC_m5_c)[2,2])


SEcol2<- rbind(SE_c_hc0, SE_c_hc1, SE_c_hc2, SE_c_hc3, SE_c_func, SE_w, SE_c_w)
cname_SEcol2 <- c("Robust Standard Errors")
rname_sEcol2 <- c("Clustered HC0", "Clustered HC1", "Clustered HC2", "Clustered HC3", "Clustered SEs fro
rownames(SEcol2) <- rname_sEcol2
colnames(SEcol2) <- cname_SEcol2

SEcol2
xtable(SEcol2)
print(xtable::xtable(SEcol2, caption = "Calculating Robust Standard Errors consider Clustering"),type =
      html.table.attributes="border=1", comment=F, floating = F)
```