Lab 5

Y. Samuel Wang

Heteroscedasticity

In this lab, we'll first show how to run a Bruesch Pagan Test and calculate the robust standard errors in R. We'll use the function `bptest` from the `lmtest` package to conduct a Bresuch Pagan test. We'll generate data twice, once from a model that is homoscedastic and once from a model that is heteroscedastic.

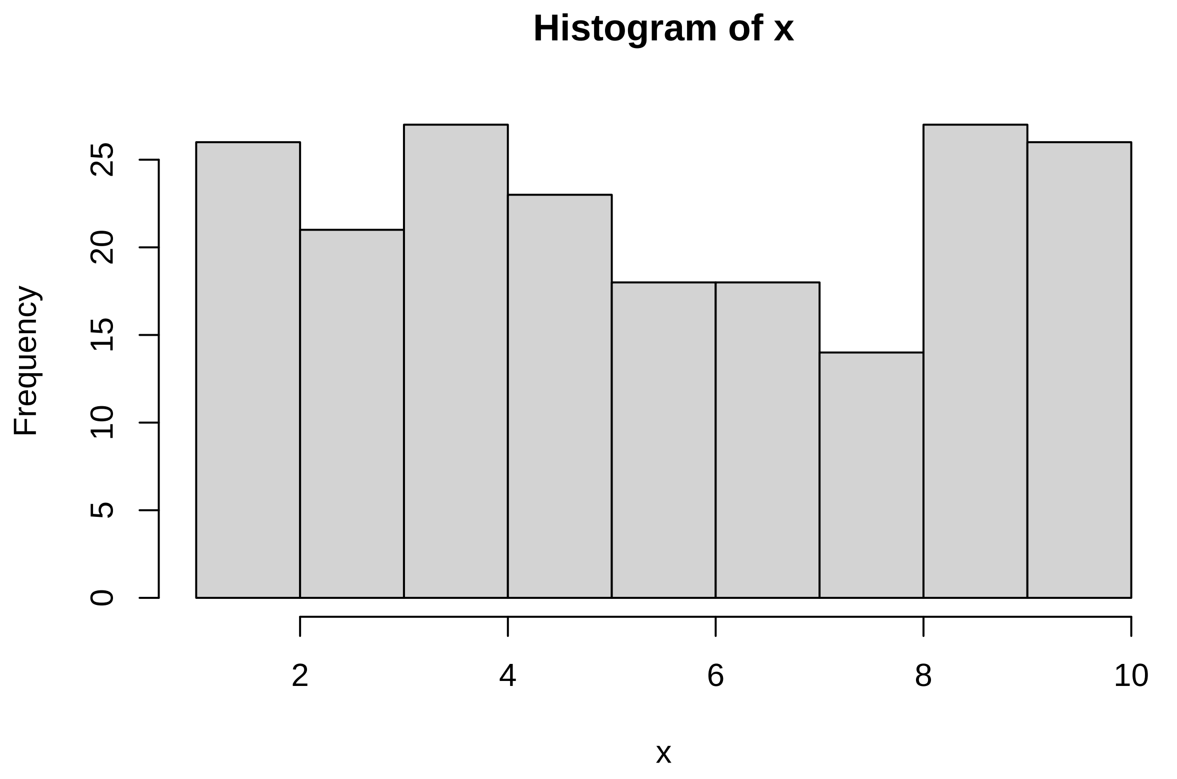
```r
#install.packages("lmtest")
library("lmtest")
```

```
## Loading required package: zoo
```

```
##
## Attaching package: 'zoo'
```

```
## The following objects are masked from 'package:base':
##
##     as.Date, as.Date.numeric
```

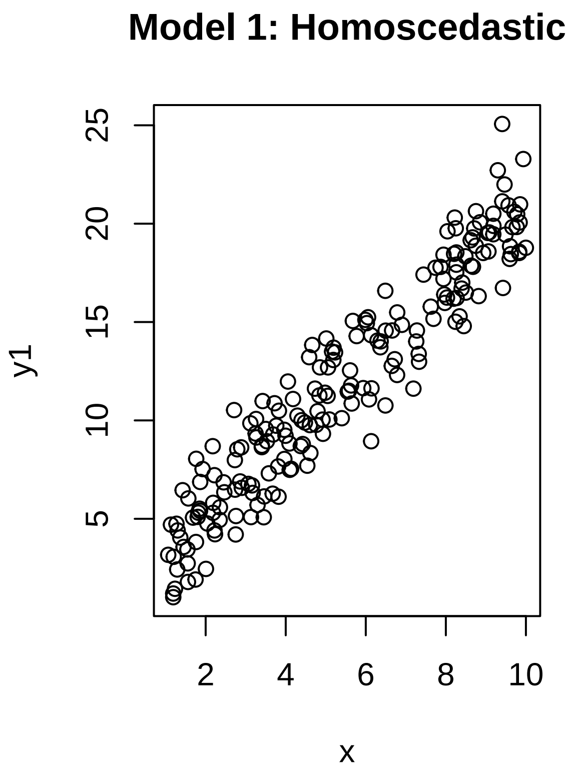
```r
n <- 200
# generate covariate X
x <- runif(n, 1, 10)
hist(x)
```

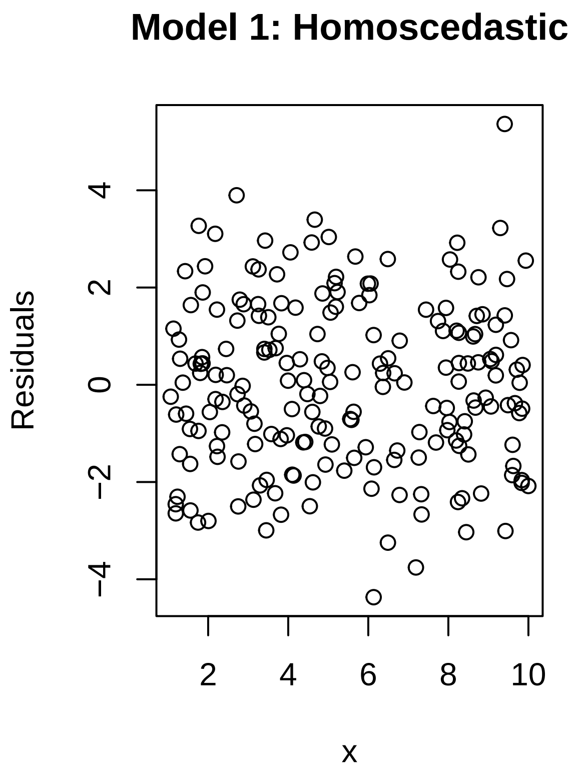


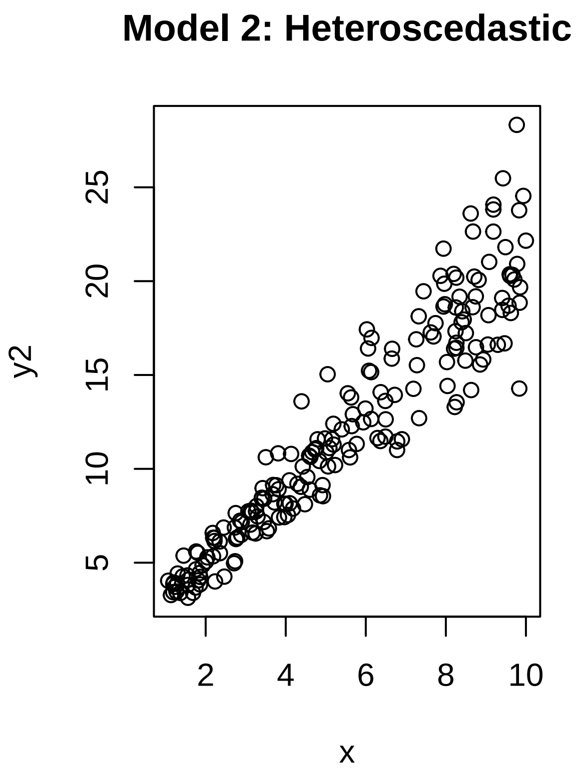
```
# generate Y2 with homoscedasticity
sd1 <- mean(x/3)
sd1
```

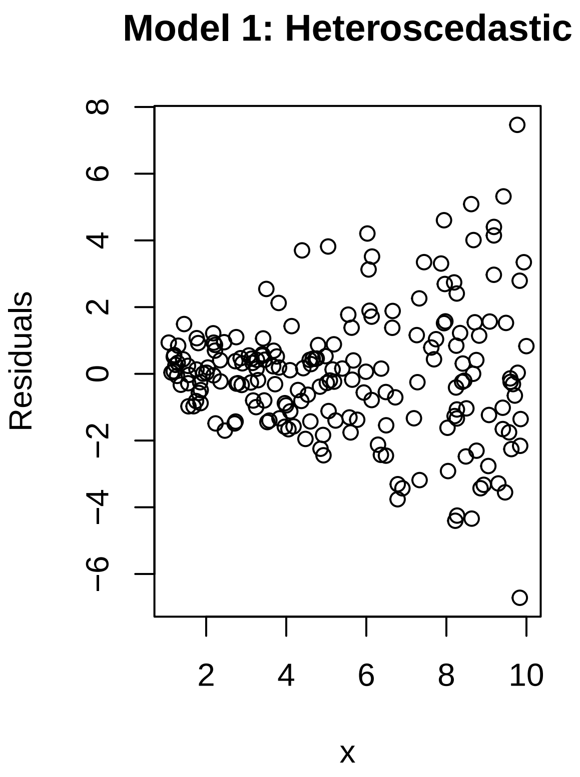
```
## [1] 1.810813
```

```
y1 <- 1 + 2 * x + rnorm(n, sd = sd1)

# generate Y2 with heteroscedasticity
y2 <- 1 + 2 * x + rnorm(n, sd = x/3)

# fit linear model
mod1 <- lm(y1 ~ x)
mod2 <- lm(y2 ~ x)
```









We can tell from the plots that the second model is heteroscedastic. But just to confirm, we can run a Bresuch Pagan test.

```r
# run breusch pagan test
# use the bptest function (in the lmtest package)
# takes the fitted linear model as an argument
bptest(mod1)
```

```
##
##  studentized Breusch-Pagan test
##
## data:  mod1
## BP = 0.024492, df = 1, p-value = 0.8756
```

```r
bptest(mod2)
```

```
##
##  studentized Breusch-Pagan test
##
## data:  mod2
## BP = 37.338, df = 1, p-value = 9.933e-10
```

The `sandwich` package calculates robust standard errors.

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

## Get model based standard errors
## vcov returns hat sigma^2 (X'X)^{-1}
## which is the estimated covariance matrix of b-hat
vcov(mod2)
```

```
##             (Intercept)            x
## (Intercept)  0.09239750 -0.013590676
## x           -0.01359068  0.002501763
```

```
# diag gets the diagonal elements of the matrix
# these elements correspond to the estimated variance of the coefficients
# We take the square root to get the standard error
sqrt(diag(vcov(mod2)))
```

```
## (Intercept)           x
##  0.30396958  0.05001762
```

```
## Get sandwich standard errors
## vcovHC (from the sandwich package) returns
# hat sigma^2 (X'X)^{-1} (X' W-hat X) (X'X)^{-1} which is
## the estimated covariance matrix of b-hat allowing for heteroscedasticity
# type = "HC3" is a specific way to estimate the W-hat matrix
# and is the most popular procedure
vcovHC(mod2, type = "HC3")
```

```
##             (Intercept)            x
## (Intercept)  0.05129183 -0.011696503
## x           -0.01169650  0.003205327
```

```
# Get the standard deviation of each of
sqrt(diag(vcovHC(mod2, type = "HC3")))
```

```
## (Intercept)           x
##  0.22647699  0.05661561
```

```
## Use the coeftest function (from the lmtest package)
# by default, the coeftest function uses the "model based" standard errors

coeftest(mod2)
```

```
##
## t test of coefficients:
##
##             Estimate Std. Error t value  Pr(>|t|)
## (Intercept) 0.929930   0.303970  3.0593  0.002526 **
## x           2.039848   0.050018 40.7826 < 2.2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
summary(mod2)
```

```
##
## Call:
## lm(formula = y2 ~ x)
```

```
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -6.7132 -1.1188  0.0195  0.8640  7.4640
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.92993    0.30397   3.059  0.00253 **
## x            2.03985    0.05002  40.783  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.927 on 198 degrees of freedom
## Multiple R-squared:  0.8936, Adjusted R-squared:  0.8931
## F-statistic:  1663 on 1 and 198 DF,  p-value: < 2.2e-16
```

```
# Create 95% confidence intervals using model based standard errors
coefci(mod2, level = .95)
```

```
##                 2.5 %   97.5 %
## (Intercept) 0.3304969 1.529364
## x           1.9412120 2.138483
```

```
# Instead of using the default model based standard errors
# we can feed a specific variance matrix
# and replace the default with the robust standard errors
coeftest(mod2, vcov. = vcovHC(mod2, type = "HC3"))
```

```
##
## t test of coefficients:
##
##             Estimate Std. Error t value  Pr(>|t|)
## (Intercept) 0.929930   0.226477  4.1061 5.884e-05 ***
## x           2.039848   0.056616 36.0298 < 2.2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
# Create 95% confidence intervals using robust standard errors
coefci(mod2, level = .95, vcov. = vcovHC(mod2, type = "HC3"))
```

```
##                 2.5 %   97.5 %
## (Intercept) 0.4833136 1.376547
## x           1.9282006 2.151495
```

Bootstrap

Bootstrapping Linear Models

Now we'll consider bootstrapping linear models as discussed in class. We'll examine the housing price data set again, and fit a model which predicts the log(price) given the log(area), bedrooms, bathrooms, whether there is a garage, and quality.

```
# install.packages("lmboot")
library("lmboot")

fileName <- url("https://raw.githubusercontent.com/ysamwang/btry6020_sp22/main/lectureData/estate.csv")
housing_data <- read.csv(fileName)
names(housing_data)
```

```
##  [1] "id"      "price"   "area"    "bed"     "bath"    "ac"      "garage"
##  [8] "pool"    "year"    "quality" "style"   "lot"     "highway"
```

```
mod <- lm(log(price) ~ log(area) + bed + bath + garage + quality,
          data = housing_data)
summary(mod)
```

```
##
## Call:
## lm(formula = log(price) ~ log(area) + bed + bath + garage + quality,
##     data = housing_data)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.45576 -0.11130 -0.01898  0.11071  0.66293
##
## Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)    7.160399   0.373898  19.151  < 2e-16 ***
## log(area)      0.695670   0.050503  13.775  < 2e-16 ***
## bed            0.001827   0.010254   0.178 0.858647
## bath           0.050975   0.013234   3.852 0.000132 ***
## garage         0.064141   0.015402   4.164 3.66e-05 ***
## qualitylow    -0.473414   0.040721 -11.626  < 2e-16 ***
## qualitymedium -0.350370   0.030252 -11.582  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.1847 on 515 degrees of freedom
## Multiple R-squared:  0.8191, Adjusted R-squared:  0.817
## F-statistic: 388.7 on 6 and 515 DF,  p-value: < 2.2e-16
```

We can now form confidence intervals for each of the parameters using a bootstrap procedure. We can see that in this case, each of the procedures we use to create confidence intervals, which should hopefully be reassuring that our model assumptions aren't too unreasonable.

```
## Gives list of parameter estimates from each empirical bootstrap
# (or paired boostrap) resample
paired.output <- paired.boot(log(price) ~ log(area) + bed + bath + garage + quality,
          data = housing_data, B = 1000)
## Gives list of parameter estimates from each wild bootstrap  resample
wild.output <- wild.boot(log(price) ~ log(area) + bed + bath + garage + quality,
          data = housing_data, B = 1000)


## Apply takes a function (FUN) and applies it to each row or column
## of a matrix
## MAR = 1, applies the function to each row
## MAR = 2, applies the function to each column

## Get standard deviation of each column (which corresponds to the bootstrap estimates)
paired.se <- apply(paired.output$bootEstParam, MAR = 2, FUN = sd)
wild.se <- apply(wild.output$bootEstParam, MAR = 2, FUN = sd)

## Get .025 and .975 quantiles of each column
paired.pct <- apply(paired.output$bootEstParam,
                    MAR = 2, FUN = quantile, prob = c(.025, .975))
wild.pct <- apply(wild.output$bootEstParam,
                  MAR = 2, FUN = quantile, prob = c(.025, .975))

## Form the confidence intervals using the normal approximation
# but SE's estimated via bootstrap
cbind(mod$coef -1.96 * paired.se, mod$coef + 1.96 * paired.se)
```

```
##                      [,1]        [,2]
## (Intercept)    6.37622692  7.94457172
## log(area)      0.58675957  0.80458074
## bed           -0.02211336  0.02576743
## bath           0.01904597  0.08290432
## garage         0.02903833  0.09924317
## qualitylow    -0.57087395 -0.37595331
## qualitymedium -0.42438700 -0.27635333
```

```
cbind(mod$coef -1.96 * wild.se, mod$coef + 1.96 * wild.se)
```

```
##                      [,1]        [,2]
## (Intercept)    6.43367520  7.88712344
## log(area)      0.59400070  0.79733961
## bed           -0.02262213  0.02627620
## bath           0.01920413  0.08274616
## garage         0.02961761  0.09866389
## qualitylow    -0.56769158 -0.37913568
## qualitymedium -0.42428972 -0.27645061
```

```
# Percentile bootstrap
t(paired.pct)
```

```
##                      2.5%       97.5%
## (Intercept)    6.37960126  7.91955084
## log(area)      0.59077841  0.80346235
## bed           -0.02299110  0.02591533
## bath           0.02267249  0.08428418
## garage         0.03160457  0.09922906
## qualitylow    -0.57434346 -0.37418222
## qualitymedium -0.42580822 -0.28034936
```

```
t(wild.pct)
```

```
##                     2.5%       97.5%
## (Intercept)    6.49399184  7.91120990
## log(area)      0.58981177  0.79111234
## bed           -0.02133519  0.02461313
## bath           0.02064398  0.08472081
## garage         0.03170418  0.09922703
## qualitylow    -0.56860088 -0.37936455
## qualitymedium -0.42063848 -0.27762555
```

```
# Model based confidence interval
coefci(mod, level = .95)
```

```
##                    2.5 %      97.5 %
## (Intercept)    6.42584727  7.89495136
## log(area)      0.59645341  0.79488690
## bed           -0.01831687  0.02197094
## bath           0.02497504  0.07697524
## garage         0.03388163  0.09439986
## qualitylow    -0.55341409 -0.39341317
## qualitymedium -0.40980286 -0.29093747
```

```
# Robust confidence intervals
coefci(mod, level = .95, vcov. = vcovHC(mod, type = "HC3"))
```

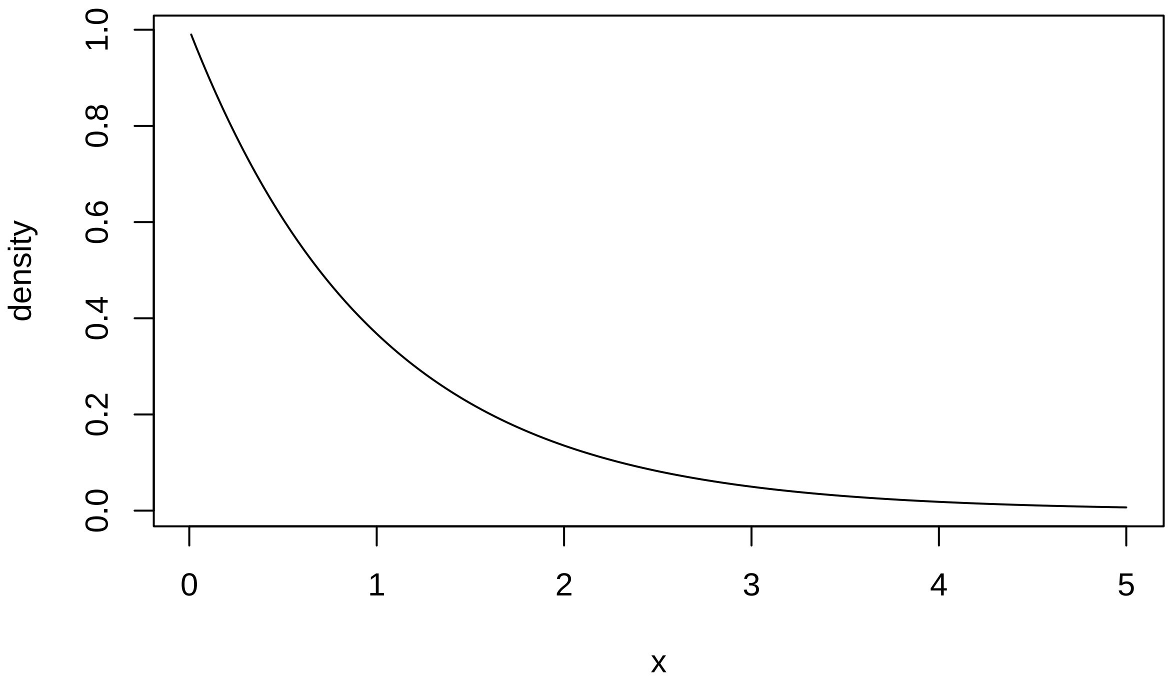
```
##                    2.5 %      97.5 %
## (Intercept)    6.35774837  7.96305026
## log(area)      0.58416815  0.80717215
## bed           -0.02377117  0.02742524
## bath           0.01777536  0.08417493
## garage         0.02891852  0.09936298
## qualitylow    -0.57206169 -0.37476557
## qualitymedium -0.42573251 -0.27500781
```

Questions

- Does it seem reasonable to use the pair (or case) bootstrap? Do the covariates seem to be fixed, or are they drawn from a random distribution?

Bootstrapping difficult quantities

Suppose we have data that is drawn from an **exponential** distribution. For the purposes of this lab, the important thing to note is that it's a skewed distribution. The density is plotted below:



If we are interested in the mean, the central limit theorem says that the sample mean of our data will be close to a normal distribution as we get more and more observations. Thus, we can form a confidence interval in the same way that you learned in BTRY 6010. However, since the data is skewed, perhaps we're more interested in the median. Unfortunately, there isn't a straightforward way to get a confidence interval for the median. So we'll use empirical bootstrap from the `boot` package.

```
library("boot")
?boot
```

```
set.seed(101)
n <- 20
## Draw n observations from an exponential distribution
X <- rexp(n)

# Function which we will feed into the boot function
# takes two arguments
# the data we observe
# indices which are drawn with replacement from 1 to n
# returns the statistic we are interested in (in this case the median) calculated
# on the bootstrapped sample
med.boot <- function(data, indices){
  return(median(data[indices]))
}

# The boot function takes in the data
# and requires a function used calculate the statistic of interest (in this
# case the median).
# R determines the number of times to resample the data
boot.output <- boot(data = X, statistic = med.boot, R = 500)

# calculate various types of 95% confidence intervals
# norm: estimates the standard error of the statistic using the bootstrap
# and plugs that estimate in to the normal confidence interval
# perc: is the percentage procedure which simply takes the quantiles of the
```

```r
# bootstrapped statistics
# bca: is a more complicated procedure which we won't cover, but has many
# theoretical and empirical advantages
ci.output <- boot.ci(boot.output, conf = .95, type = c("norm", "perc", "bca"))

# includes negative values
# Element 1, is just the confidence level
# the confidence interval is in elements 4/5
ci.output$normal
```

```
##      conf
## [1,] 0.95 -0.1790461 1.057167
```

```r
# confidence interval from percentile procedure
# don't worry about elements 1-3, the confidence interval is in elements 4/5
ci.output$perc[c(4,5)]
```

```
## [1] 0.3170879 1.2848651
```

```r
# bca confidence interval
# don't worry about elements 1-3, the confidence interval is in elements 4/5
ci.output$bca[c(4,5)]
```

```
## [1] 0.3170879 1.2785064
```

So we can form confidence intervals for the median, but do they work like we think they should? The median for the exponential distribution is log(2). Let's repeat the procedure many different times and see if we actually do include the median 95% of the time.

```r
# number of times we'll repeat the experiment
sim.size <- 500

n <- 20

# where we will record relevant statistics

rec <- matrix(0, sim.size, 6)
colnames(rec) <- c("Normal Coverage", "Percentile Coverage", "Bca Coverage",
                   "Normal length", "Percentile length", "Bca length")
for(i in 1:sim.size){
  # draw fresh data
  X <- rexp(n)

  # calculate the 3 different confidence intervals
  boot.output <- boot(data = X, statistic = med.boot, R = 500)
  ci.output <- boot.ci(boot.output, conf = .95, type = c("norm", "perc", "bca"))

  # check if log(2) is in the normal CI
  rec[i, 1] <- ci.output$normal[2] < log(2) & ci.output$normal[3] > log(2)
  # record size of CI
  rec[i, 4] <- ci.output$normal[3] - ci.output$normal[2]
  # check if log(2) is in the percentile CI
  rec[i, 2] <- ci.output$perc[4] < log(2) & ci.output$perc[5] > log(2)
  rec[i, 5] <- ci.output$perc[5] - ci.output$perc[4]
  # check if log(2) is in the percentile CI
  rec[i, 3] <- ci.output$bca[4] < log(2) & ci.output$bca[5] > log(2)
  rec[i, 6] <- ci.output$bca[5] - ci.output$bca[4]

}



# first three columns indicate whether the normal/percentile/bca CI's
# contained the true median (we want this to be close to .95)
# last three columns are the length of the normal/percentile/bca CI's
# We take the mean of the columns and round to 4 digits
round(colMeans(rec), 4)
```

```
##     Normal Coverage Percentile Coverage        Bca Coverage       Normal length
##              0.9260              0.9420              0.9480              0.9264
##   Percentile length          Bca length
##              0.8946              0.8647
```

Questions

- How do the procedures compare? Is there a particular procedure which seems to do the best in terms of coverage?
- If different procedures can give you a CI with exactly 95% coverage, how else might you decide between procedures?
- If you increase the number of samples (maybe try $n = 30$ or 50), how does the coverage change? How does the length of the CI's change?

Potential answers to discussion uestions

- Does it seem reasonable to use the pair (or case) bootstrap? Do the covariates seem to be fixed, or are they drawn from a random distribution?
  - The pairs bootstrap seems reasonable in this case because the data is observational and the covariate levels are not set. If we were to go out and get a new data set, it's likely that we would see different covariates for new houses.
- If different procedures can give you a CI with exactly 95% coverage, how else might you decide between procedures?
  - If two different procedures both give the correct covareage rate, we would probably prefer the procedure that gives shorter intervals. This would mean that it's more precise and includes the "correct" value the right proportion of the time, but doesn't include as many "incorrect" values. ?