Lab 6

In this lab, we'll first show how to run a Bruesch Pagan Test to check for heteroscedasticity. We'll also calculate the robust standard errors in R and explore when using the robust standard errors is advantageous. Finally, we'll briefly introduce the idea of the bootstrap, which we'll explore more in class on Wednesday.

Heteroscedasticity

Breusch Pagan Test

To see how to run the BP test in R, we'll generate data twice. Once from a model that is homoscedastic and once from a model that is heteroscedastic.

```
#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
```

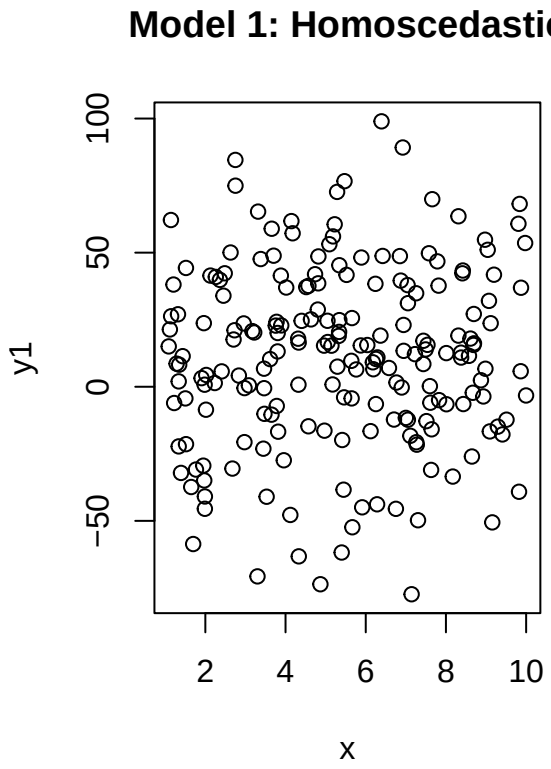
```
n <- 200
# generate covariate X
x <- runif(n, 1, 10)

# generate Y1 with homoscedasticity
sd1 <- mean(x^2)
sd1
```

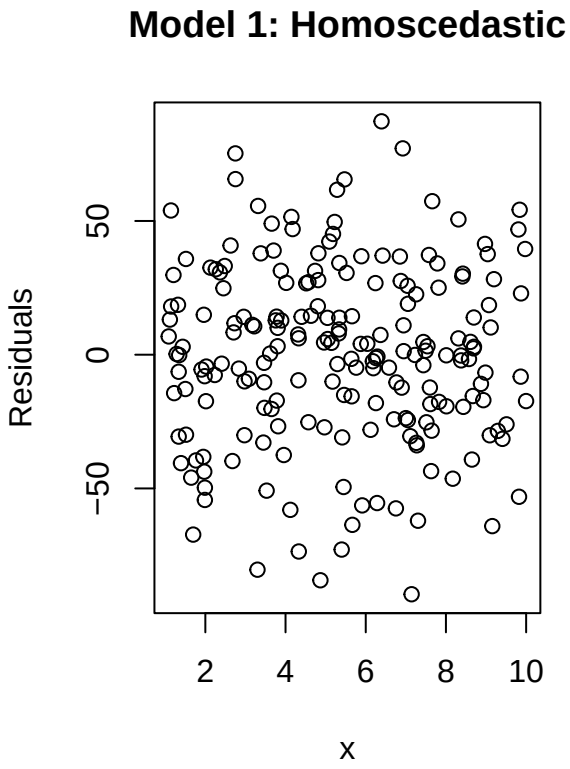
```
## [1] 34.96661
```

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

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

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

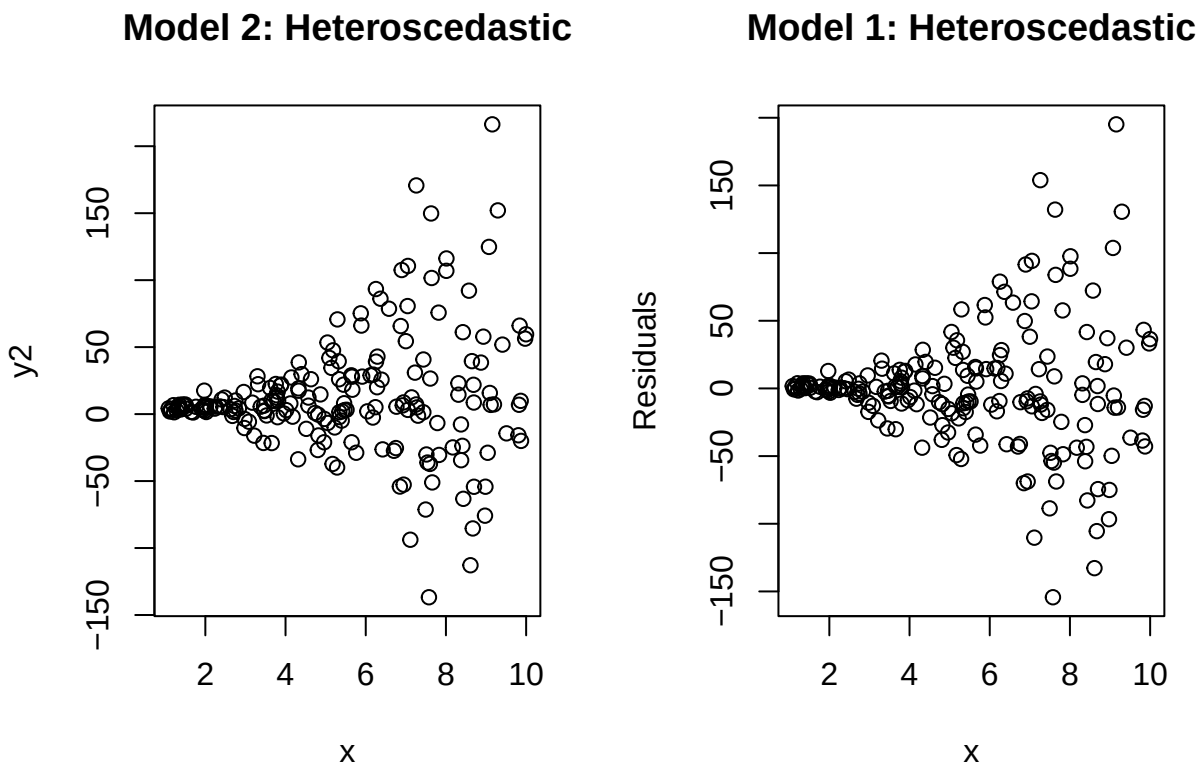
We can examine the residuals of each data set

```
par(mfrow = c(1,2))
plot(x, y1, main = "Model 1: Homoscedastic")
plot(x, mod1$res, main = "Model 1: Homoscedastic", ylab = "Residuals")
```



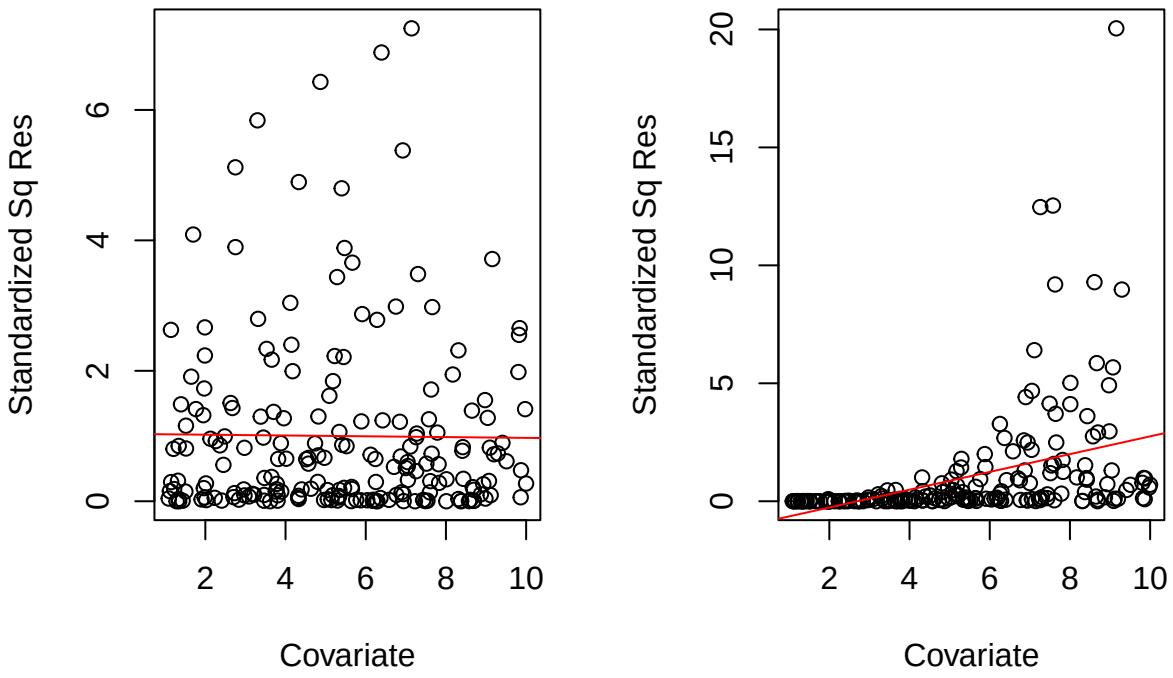



```
par(mfrow = c(1,2))
plot(x, y2, main = "Model 2: Heteroscedastic")
plot(x, mod2$res, main = "Model 1: Heteroscedastic", ylab = "Residuals")
```



We can tell from the plots that the second model is heteroscedastic. But just to confirm, we can run a Bresuch Pagan test. Recall that the BP test regresses the standardized squared residuals onto the covariates and checks whether all coefficients are 0. In this case, we only have one covariate, so it's a bit easier to visualize what exactly we're checking.

```
## Standardized Squared residuals
stSqRes1 <- mod1$residuals^2 / mean(mod1$residuals^2)
stSqRes2 <- mod2$residuals^2 / mean(mod2$residuals^2)

par(mfrow = c(1,2))
plot(x, stSqRes1, ylab = "Standardized Sq Res", xlab = "Covariate")
abline(a = lm(stSqRes1~x)$coef[1], b = lm(stSqRes1~x)$coef[2], col = "red")
plot(x, stSqRes2, ylab = "Standardized Sq Res", xlab = "Covariate")
abline(a = lm(stSqRes2~x)$coef[1], b = lm(stSqRes2~x)$coef[2], col = "red")
```



Indeed, we see that the BP test fails to reject the null hypothesis for the first setting where the data truly is homoscedastic. However, we do reject the null hypothesis for the 2nd settting where the data is heteroscedastic.

```
# 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 = 0.024974, df = 1, p-value = 0.8744
```

```
bptest(mod2)
```

```
##
##  studentized Breusch-Pagan test
##
## data:  mod2
## BP = 31.3, df = 1, p-value = 2.211e-08
```

What goes wrong?

Let's again, remind ourselves of what goes wrong when the errors are heteroscedastic. We let

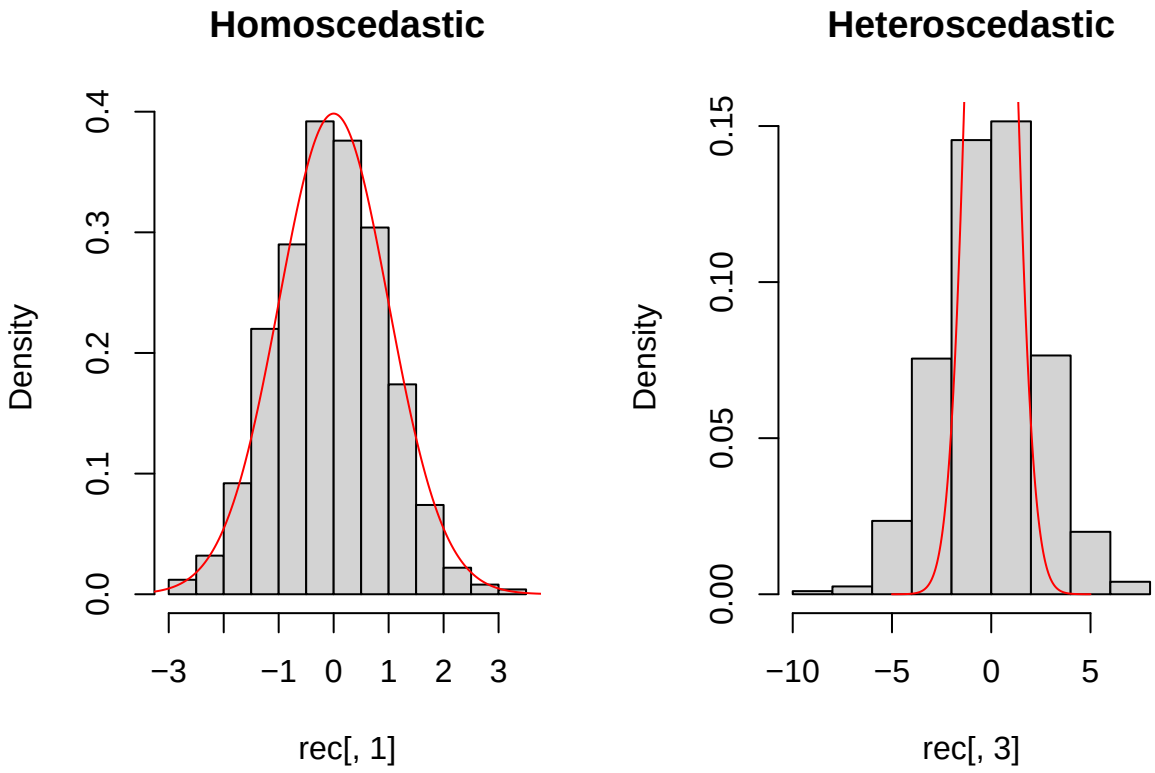
$$Y_i = bX_i + \varepsilon_i,$$

and set $b = 0$ at first. Then the null hypothesis is true, and we ought to be able to describe distribution of the test statistic

$$t = \frac{\hat{b}}{\widehat{\text{var}}(\hat{b})}$$

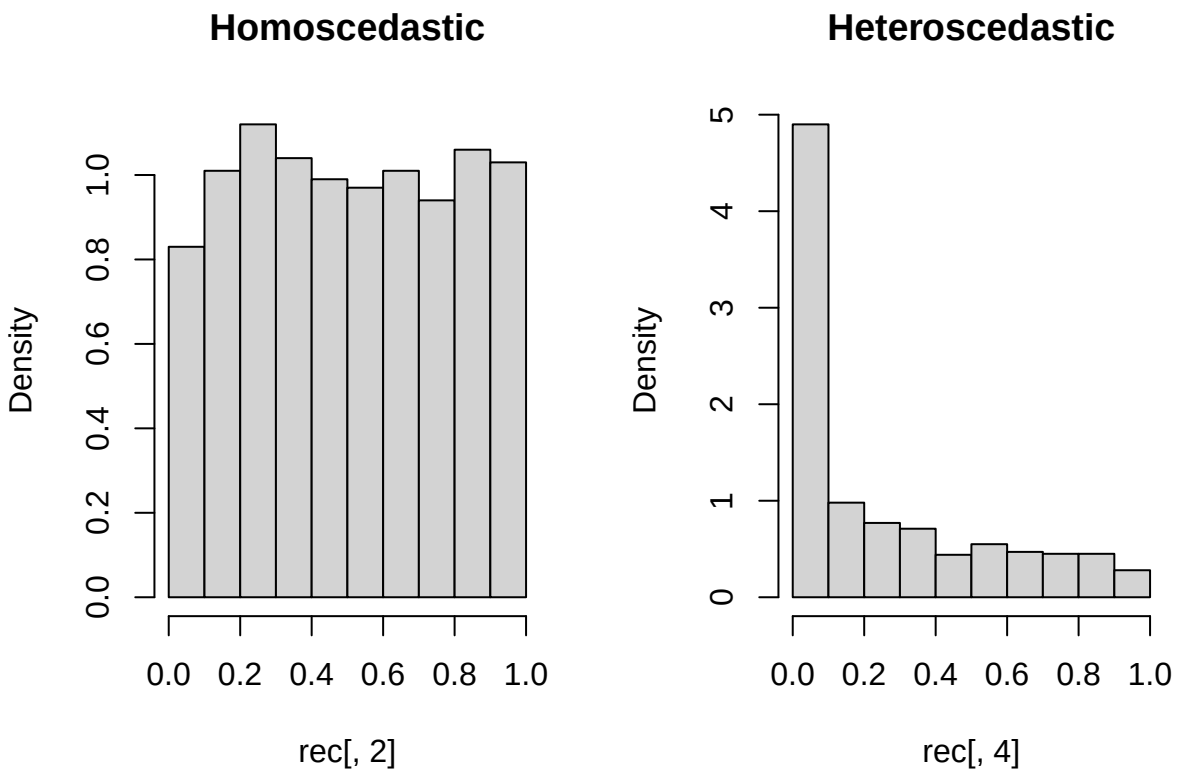
by a T_{n-2}. Below, we simulate data and plot the empirical distribution as well as the theoretical distribution in red.

```
sim.size <- 1000

b <- 0
n <- 200
# generate covariate X

rec <- matrix(0, sim.size, 4)

for(i in 1:sim.size){
  x <- rgamma(n, 1, 1)
  # generate Y2 with homoscedasticity
  sd1 <- mean(x)
  y1 <- 1 + b * x + rnorm(n, sd = sd1)

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


  mod1 <- lm(y1 ~ x)
  mod2 <- lm(y2 ~ x)
  # fit linear model
  output1 <- summary(mod1)$coeff
  output2 <- summary(mod2)$coeff


  rec[i, ] <- c(output1[2,3:4], output2[2,3:4])
}

x <- seq(-5, 5, by = .01)
par(mfrow = c(1,2))
hist(rec[, 1], freq = F, main = "Homoscedastic")
lines(x, dt(x, df = n - 2), col = "red")
hist(rec[, 3], freq = F, main = "Heteroscedastic")
lines(x, dt(x, df = n - 2), col = "red")
```



We can also look at the distribution of the resulting p-values.

```
par(mfrow = c(1,2))
hist(rec[, 2], freq = F, main = "Homoscedastic")
hist(rec[, 4], freq = F, main = "Heteroscedastic")
```



Questions

- Does the distribution of the test statistic and p-values under the homoscedastic setting look like what you'd expect? Why or why not?
- Does the distribution of test statistic and the p-values under the heteroscedastic setting look like what you'd expect? Why or why not?
- When the data is heteroscadstic, but we don't properly account for that, what type of error are we likely making?
- Check the proportion of times that error is made under the homoscedastic setting, and compare it to the proportion of times the error is made under the heteroscedastic setting.

Robust Standard errors

To deal with the problem of heteroscedasticity, we can use the `sandwich` package to calculate 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)  0.011312464 -0.005799291
## x           -0.005799291  0.005910294
```

```r
# 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
##  0.10636007  0.07687844
```

```r
# Check to see that this is the same as
summary(mod2)
```

```
##
## Call:
## lm(formula = y2 ~ x)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -4.8994 -0.3374 -0.0054  0.3395  3.7299
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  1.02145    0.10636   9.604   <2e-16 ***
## x            0.11978    0.07688   1.558    0.121
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.06 on 198 degrees of freedom
## Multiple R-squared:  0.01211,    Adjusted R-squared:  0.007122
## F-statistic: 2.427 on 1 and 198 DF,  p-value: 0.1208
```

```r
## 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)  0.005442637 -0.005823194
## x           -0.005823194  0.012173380
```

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

```
## (Intercept)           x
##  0.07377423  0.11033304
```

```r
# Check to see that this is NOT the same as
summary(mod2)
```

```
##
## Call:
## lm(formula = y2 ~ x)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
```

```
## -4.8994 -0.3374 -0.0054  0.3395  3.7299
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  1.02145    0.10636   9.604   <2e-16 ***
## x            0.11978    0.07688   1.558    0.121
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.06 on 198 degrees of freedom
## Multiple R-squared:  0.01211,    Adjusted R-squared:  0.007122
## F-statistic: 2.427 on 1 and 198 DF,  p-value: 0.1208
```

```
## 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) 1.021448   0.106360  9.6037   <2e-16 ***
## x           0.119778   0.076878  1.5580   0.1208
## ---
## 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
## -4.8994 -0.3374 -0.0054  0.3395  3.7299
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  1.02145    0.10636   9.604   <2e-16 ***
## x            0.11978    0.07688   1.558    0.121
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.06 on 198 degrees of freedom
## Multiple R-squared:  0.01211,    Adjusted R-squared:  0.007122
## F-statistic: 2.427 on 1 and 198 DF,  p-value: 0.1208
```

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

```
##                   2.5 %    97.5 %
## (Intercept)  0.81170390 1.2311917
## x           -0.03182778 0.2713835
```

```
# Instead of using the default model based standard errors
# we can feed a specific variance matrix
```

```r
# and replace the default with the robust standard errors
coeftest(mod2, vcov. = vcovHC(mod2, type = "HC3"))
```

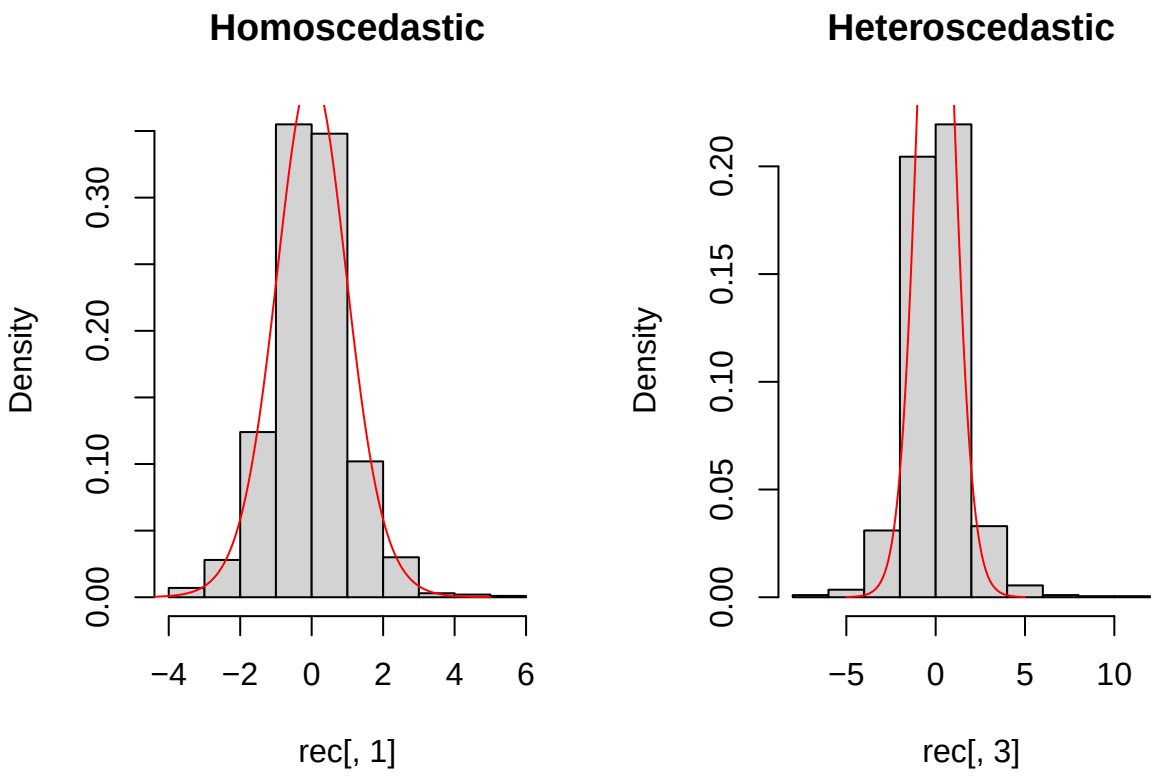
```
##
## t test of coefficients:
##
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 1.021448   0.073774 13.8456   <2e-16 ***
## x           0.119778   0.110333  1.0856    0.279
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

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

```
##                   2.5 %    97.5 %
## (Intercept)  0.87596374 1.1669319
## x           -0.09780084 0.3373565
```

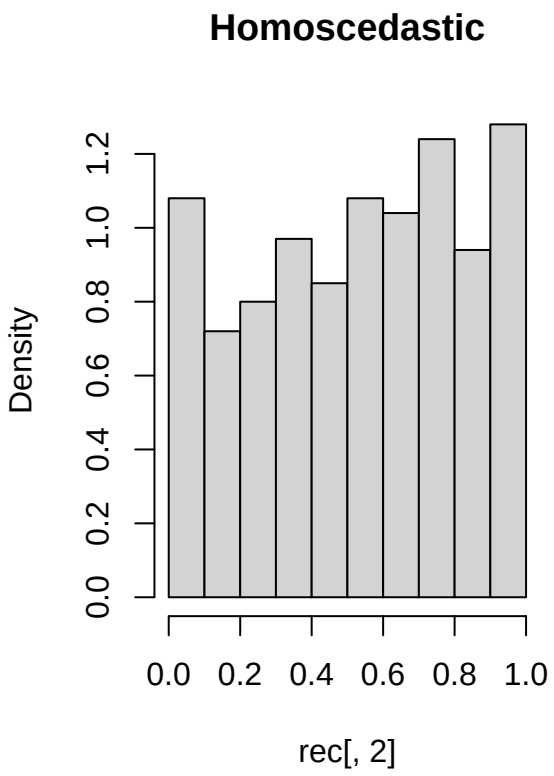
Now let's repeat everything, but set $n = 20$ and use the sandwich standard errors instead of the model based standard errors.

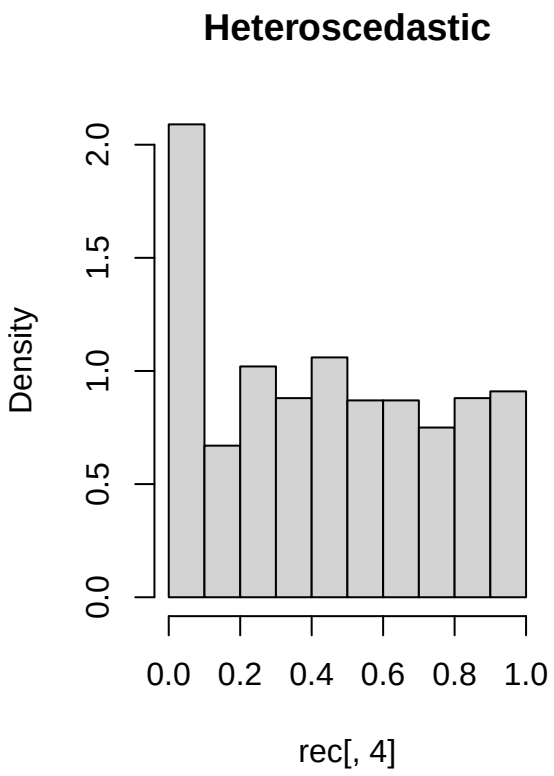
```
sim.size <- 1000

b <- 0
n <- 20
# generate covariate X

rec <- matrix(0, sim.size, 4)

for(i in 1:sim.size){
  x <- rgamma(n, 1, 1)
  # generate Y2 with homoscedasticity
  sd1 <- mean(x)
  y1 <- 1 + b * x + rnorm(n, sd = sd1)

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

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


  rec[i, ] <- c(  coeftest(mod1, vcov. = vcovHC(mod1, type = "HC3"))[2,3:4],
                  coeftest(mod2, vcov. = vcovHC(mod2, type = "HC3"))[2,3:4])
}

x <- seq(-5, 5, by = .01)
par(mfrow = c(1,2))
hist(rec[, 1], freq = F, main = "Homoscedastic")
lines(x, dt(x, df = n - 2), col = "red")
hist(rec[, 3], freq = F, main = "Heteroscedastic")
lines(x, dt(x, df = n - 2), col = "red")
```



We can also look at the distribution of the resulting p-values.

```
par(mfrow = c(1,2))
hist(rec[, 2], freq = F, main = "Homoscedastic")
hist(rec[, 4], freq = F, main = "Heteroscedastic")
```





Questions

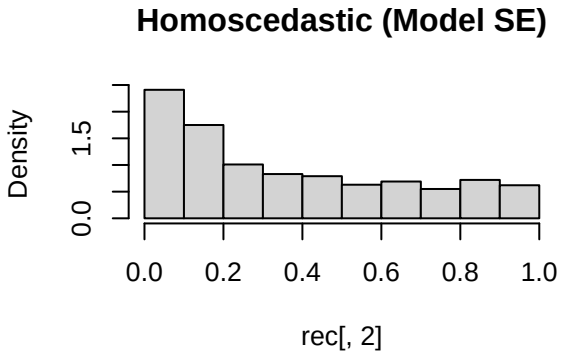
- Does the distribution of the test statistic and p-values under the homoscedastic setting look like what you'd expect? Why or why not?
- Does the distribution of test statistic and the p-values under the heteroscedastic setting look like what you'd expect? Why or why not?
- When the data is heteroscadstic, but we don't properly account for that, what type of error are we likely making?
- Check the proportion of times that error is made under the homoscedastic setting, and compare it to the proportion of times the error is made under the heteroscedastic setting.
- Now try the same simulation again, but set $n = 200$. What seems to change?

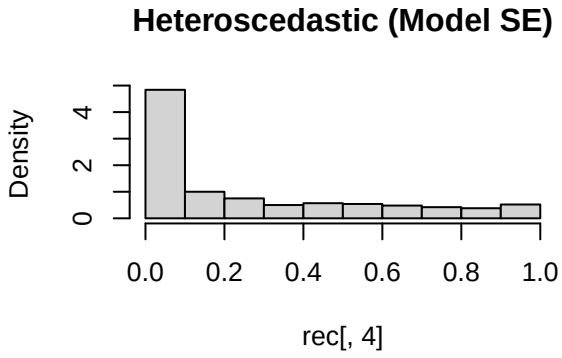
Finally, let's repeat things one more time, but set $b = 1/4$ so that null hypothesis $H_0 : b = 0$ is false. Now let's repeat everything again.

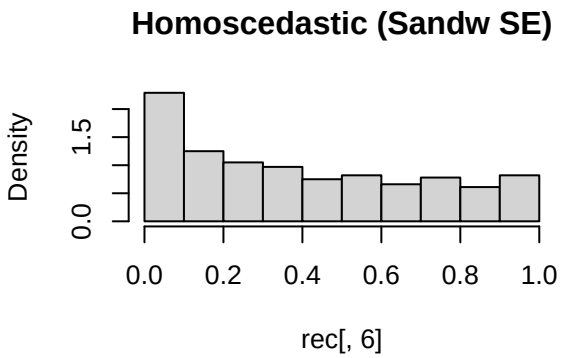
```
sim.size <- 1000

b <- 1/4
n <- 20
# generate covariate X

rec <- matrix(0, sim.size, 8)

for(i in 1:sim.size){
  x <- rgamma(n, 1, 1)
  # generate Y2 with homoscedasticity
  sd1 <- mean(x)
  y1 <- 1 + b * x + rnorm(n, sd = sd1)

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

  # fit linear model
  mod1 <- lm(y1 ~ x)
  mod2 <- lm(y2 ~ x)
    output1 <- summary(mod1)$coeff
  output2 <- summary(mod2)$coeff


  rec[i, ] <- c(output1[2,3:4],
                output2[2,3:4],
                coeftest(mod1, vcov. = vcovHC(mod1, type = "HC3"))[2,3:4],
                coeftest(mod2, vcov. = vcovHC(mod2, type = "HC3"))[2,3:4])
}
```

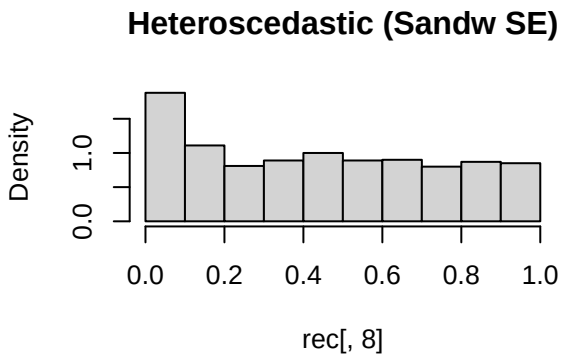
We can plot the p-values which result from each setting.

```
par(mfrow = c(2,2))
hist(rec[, 2], freq = F, main = "Homoscedastic (Model SE)")
hist(rec[, 4], freq = F, main = "Heteroscedastic (Model SE)")
hist(rec[, 6], freq = F, main = "Homoscedastic (Sandw SE)")
hist(rec[, 8], freq = F, main = "Heteroscedastic (Sandw SE)")
```









Questions

- In every setting, the null hypothesis is actually not true. What type of error might we commit here?
- When the true model is homoscedastic, does using the model based SE's or the homoscedastic SE's result in more errors?
- When the true model is heteroscedastic, does using the model based SE's or the homoscedastic SE's result in more errors?
- In the heteroscedastic case, it seems that using the model based SE's is better, because we reject the null hypothesis more often. However, this is not the whole story, because we end up not controlling the proportion of Type I errors. Explain why this is not typically a tradeoff worth making.

Housing Data

Recall the housing data we've used previously.

```
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"
```

Questions

- Fit the model

$$\log(price) = b_0 + b_1 \times \log(area) + b_2 \times \log(lot) + b_3 \times bed + b_4 \times bath + \varepsilon,$$

  and test whether heteroscedasticity can be detected or not.
- Then, test the null hypothesis

$$H_0 : b_3 = 0 \qquad \text{vs} \qquad H_A : b_3 \neq 0$$

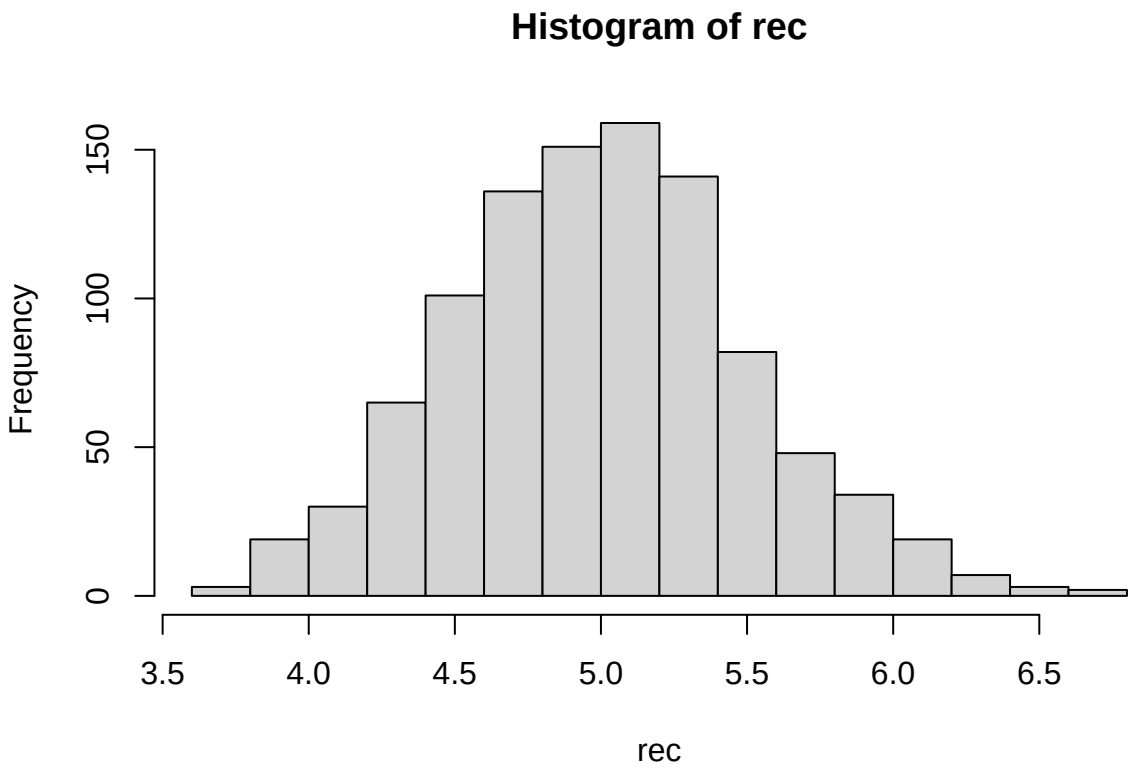
  using both model based standard errors and robust standard errors.
- How do the conclusions differ? What would you conclude if you were trying to publish a paper with this data?

Bootstrap

On Wednesday, we'll talk about the bootstrap for linear models in class. But as a headstart, we'll consider the bootstrap in less complicated settings first. So far, we have used lots of simulations to approximate sampling distributions. This involves drawing new data from a particular probability model; however, in reality, we typically do not know the probability model, so this approach is infeasible. The bootstrap provides a way to resample our data to approximate the sampling distribution.

In particular, we will look out a way to approximate the sampling distribution of the sample median. If we calculate the sample mean of $n = 50$ draws from a gamma distribution, what does the sampling distribution look like? We can simulate this as before.

```
sim.size <- 1000
n <- 20
rec <- rep(0, sim.size)

for(i in 1:sim.size){
  rec[i] <- mean(rgamma(n, 5, 1))
}
hist(rec)
```

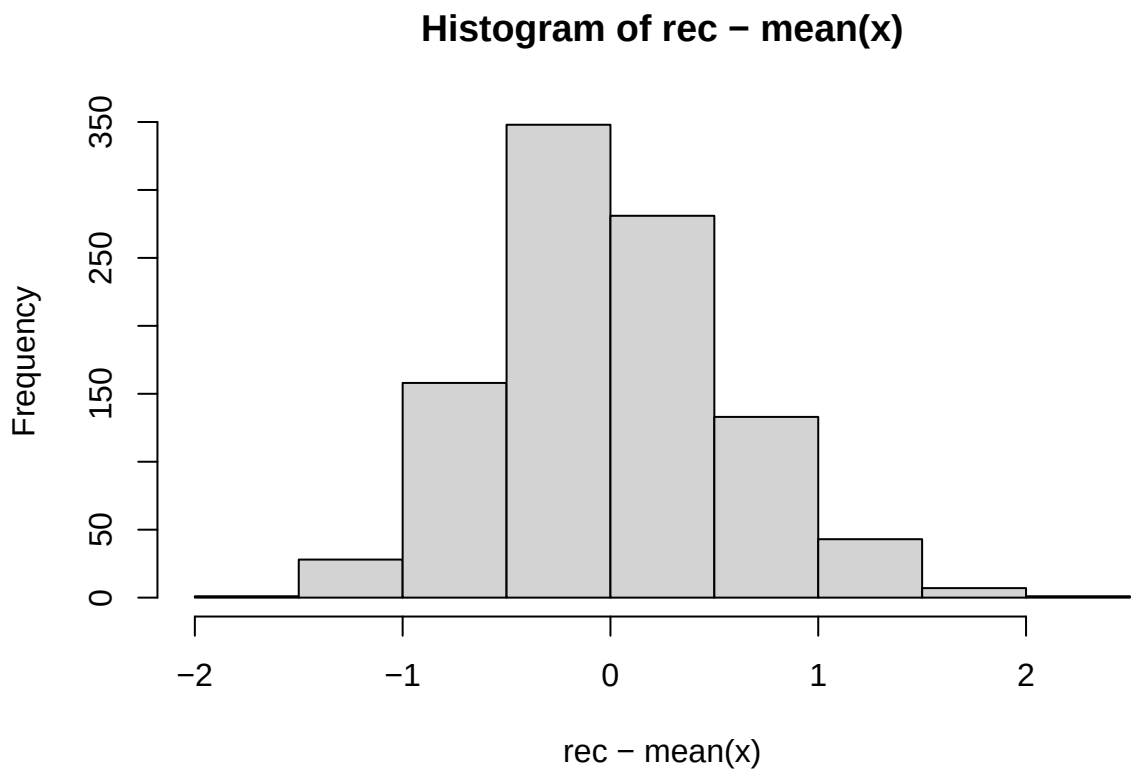


But when I don't know that the underlying data comes from a gamma distribution, what I can do is resample points from my observed data, and use that as an approximation as redrawing totally new data. In particular, we can use the `sample` function. We set `replace = T` so that we may end up including the same original point multiple times in our new bootstrapped sample.

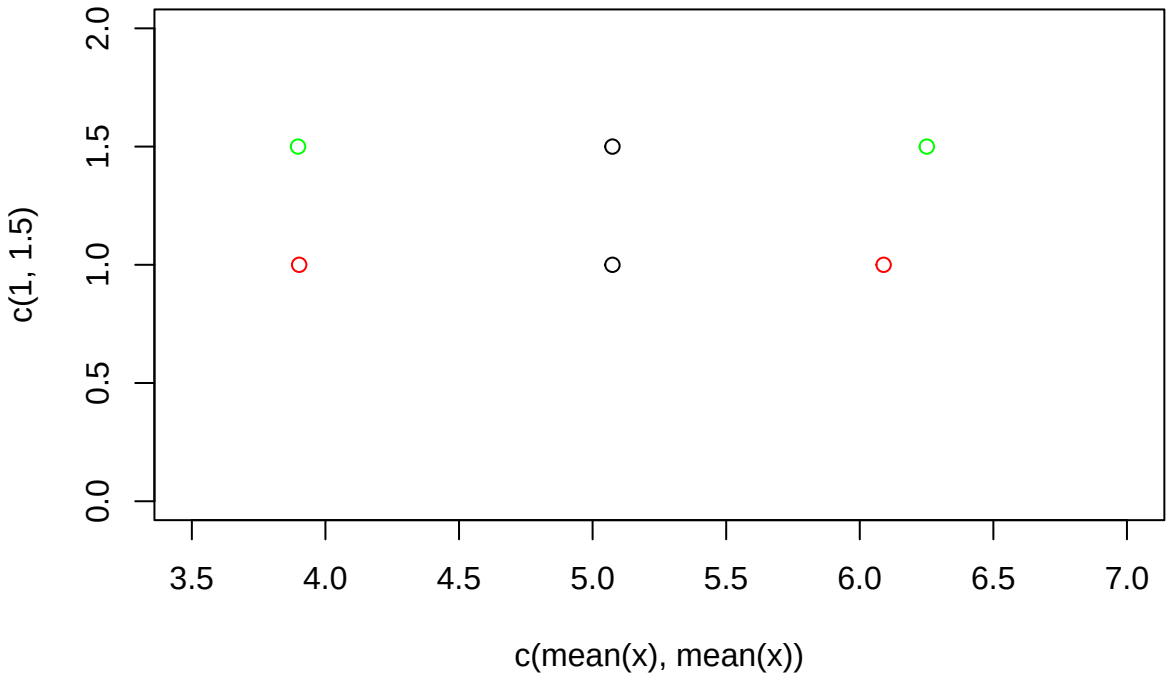
```
x <- rgamma(n, 1, 1)
newX <- sample(x, replace = T)
```

Now, let's approximate the sampling distribution of the mean, by "redrawing" new data from our actual observed set of data.

```
sim.size <- 1000

n <- 20
rec <- rep(0, sim.size)
x <- rgamma(n, 5, 1)

for(i in 1:sim.size){
  rec[i] <- mean(sample(x, replace = T) )
}

z <- quantile(rec - mean(x), c(.025, .975))
cutoff <- qt(.025, df = 20-1, lower = T)


hist(rec- mean(x))
```




```
plot(c(mean(x), mean(x)), c(1, 1.5), xlim = c(3.5,7), ylim = c(0, 2))
points(mean(x) - z, c(1,1), col = "red")
points(c(mean(x) - qt(.025, df = 20-1, lower = T) * sd(rec),
         mean(x) + qt(.025, df = 20-1, lower = T) * sd(rec)), c(1.5,1.5), col = "green")
```



Questions

- Does the approximation using the bootstrap samples look reasonably similar to the case where we got the true sampling distribution?
- Simulations are a very helpful tool for approximating things that are hard to calculate, but they require we know a probability model? Can you think of any examples where bootstrapping might be helpful when we don't know the probability model and simulations aren't possible?