Consequences of Heteroscedasticity for OLS regression

Andrija Djurovic*

*www.linkedin.com/in/andrija-djurovic

One assumption of OLS regression is that residuals exhibit homoscedasticity, implying constant variance. When this variance is not constant, practitioners identify a heteroscedasticity problem in the OLS regression. While opinions on the significance of the normality assumption vary, a consensus on heteroscedasticity leads to unbiased but inefficient estimates. Essentially, the minimal variance of the estimates is compromised, rendering inferences from OLS unreliable.

The following simulation illustrates the potential consequences of heteroscedasticity, examining efficiency through the standard error of estimates and reporting the associated power.

Starting in `R`, the data for OLS regression is simulated with two independent variables (`x1` and `x2`) and a dependent variable (`y`) drawn from the standard normal distribution. The simulated estimates for `x1` and `x2` are set at `0.85` and `-0.65`.

```
#simulation parameters
n <- 45               #number of observations
iv <- 2               #number of independent variables

#simulate the x
set.seed(123)
x <- replicate(n = iv,
               expr = rnorm(n = n)
               )
x <- data.frame(x)
names(x) <- paste0("x", 1:iv)

#true regression parameters
intercept <- 0
beta1 <- 0.85
beta2 <- -0.65

#simulate the target
y <- intercept + beta1*x$x1 + beta2*x$x2

#store the inputs into data frame
db <- data.frame(y = y, x)
```

Now that we have generated the data above, we can proceed with the simulation and evaluate the implications of heteroscedastic errors.

```
#------------------------run the simulations-----------------------#
#mc parameters
```

```r
alpha <- 0.05         #significance level
xw <- c(0.60, 0.35)   #x contribution to the heteroscedastic errors
B <- 1000             #number of mc simulations

#empty list to store the results (heteroscedastic residuals)
res.unw <- matrix(data = NA,
                  nrow = B,
                  ncol = 4)
res.unw <- data.frame(res.unw)
names(res.unw) <- c("b1", "b2", "b1.pval", "b2.pval")
#empty list to store the results of the weighted regression
res.wts <- res.unw

for   (i in 1:B) {
       #random seed
       set.seed(i * 2)

       #simulate the (centered) heteroscedastic errors
       error.i <- rnorm(n = n,
                        mean = 0,
                        sd = exp(xw[1]*db$x1 + xw[2]*db$x2))
       error.i <- error.i - mean(error.i)

       #utilize the weights to account for heteroscedasticity
       resid.i <- log(error.i^2)
       h.i <- exp(lm(formula = resid.i ~ db$x1 + db$x2)$fitted.values)
       weights.i <- 1 / h.i

       #simulate the target
       db$y.sim <- db$y + error.i

       #run the regressions
       lr.unw <- lm(formula = y.sim ~ x1 + x2,
                    data = db)
       lr.wts <- lm(formula = y.sim ~ x1 + x2,
                    weights = weights.i,
                    data = db)

       #store the simulation results
```

```
        res.unw[i, ] <- summary(lr.unw)$coefficients[2:3, c(1, 4)]
        res.wts[i, ] <- summary(lr.wts)$coefficients[2:3, c(1, 4)]
        }
```

Let's compare the true betas with those obtained from the regression with heteroscedastic errors (unweighted) and those derived from the regression where errors are modeled (weighted).

```
#confidence intervals
#true betas
c("beta1" = beta1, "beta2" = beta2)
```

```
## beta1 beta2
##  0.85 -0.65
```

```
#unweighted betas
b12.ci.unw <- sapply(X = res.unw[, c("b1", "b2")],
                     FUN = function(x) {
                           quantile(x = x,
                                    probs = c(alpha / 2, 0.5, 1 - alpha/2))
                            })
b12.ci.unw
```

```
##              b1         b2
## 2.5%  0.2081474 -1.2032158
## 50%   0.8413577 -0.6470709
## 97.5% 1.5091026 -0.1008995
```

```
#weighted betas
b12.ci.wts <- sapply(X = res.wts[, c("b1", "b2")],
                     FUN = function(x) {
                           quantile(x = x,
                                    probs = c(alpha / 2, 0.5, 1 - alpha/2))
                            })
b12.ci.wts
```

```
##              b1         b2
## 2.5%  0.5531600 -0.9935363
## 50%   0.8521381 -0.6535496
## 97.5% 1.1462048 -0.3274951
```

Lastly, we can examine the power in both approaches and observe the enhancement, particularly in the case of the weighted regression.

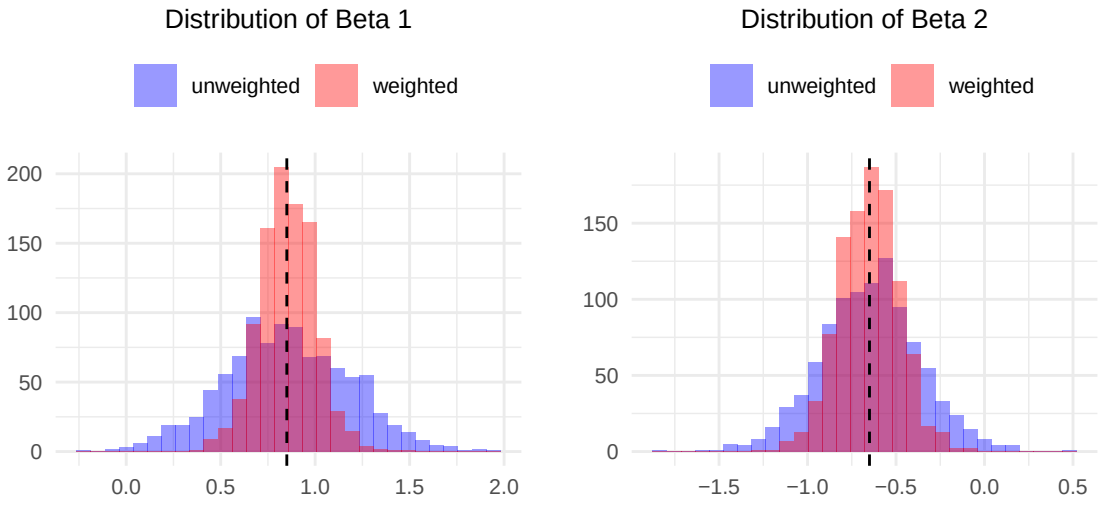
```r
#power - heteroscedastic errors
sapply(X = res.unw[, c("b1.pval", "b2.pval")],
       FUN = function(x) mean(x < alpha))
```

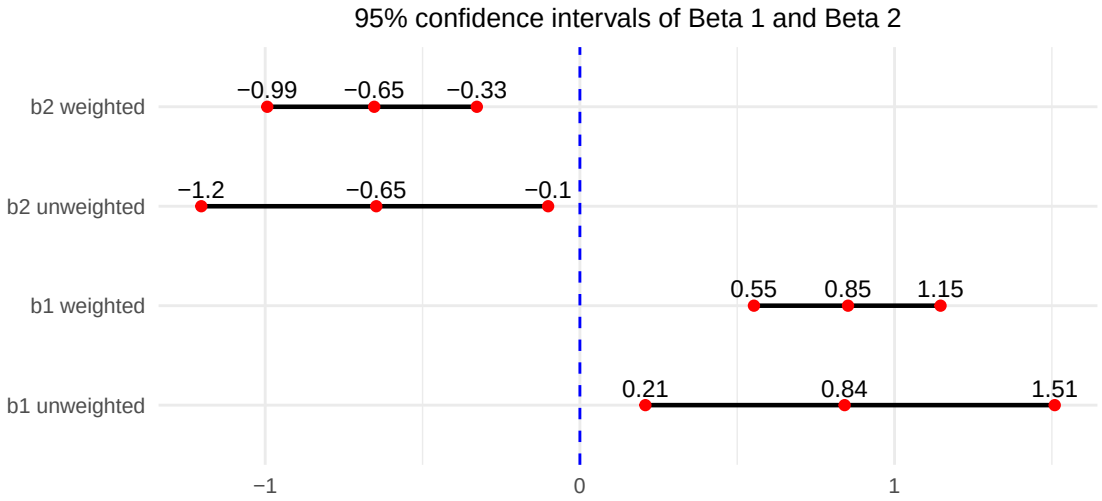
```
## b1.pval b2.pval
##   0.872   0.659
```

```r
#power - heteroscedastic errors modeled
sapply(X = res.wts[, c("b1.pval", "b2.pval")],
       FUN = function(x) mean(x < alpha))
```

```
## b1.pval b2.pval
##    1.00    0.97
```

To better comprehend the findings above, let's visually present the results.





Towards the conclusion of the exercise, we will export the simulated data to replicate the process in Python.

```
#export the db to replicate the simulation in python
write.csv(x = db[, -ncol(db)],
          file = "db.csv",
          row.names = FALSE)
```

Now, let's code the same simulation in Python.

```
import numpy as np
import pandas as pd
import statsmodels.api as sm
import statsmodels.formula.api as smf

#import db
db = pd.read_csv("db.csv")

#mc parameters
alpha = 0.05
B = 1000
xw = np.array([0.60, 0.35])

#create an empty dataframe to store results
columns = ["b1", "b2", "b1.pval", "b2.pval"]
res_unw = pd.DataFrame(index = range(B),
                       columns = columns,
                       dtype = float)
res_wts = res_unw.copy()

#------------------------run the simulations-------------------------#

for i in range(B):
    #random seed
    np.random.seed((i + 1) * 2)

    #simulate the (centered) heteroscedastic errors
    error_i = np.random.normal(loc = 0,
                               scale = np.exp(xw[0] * db['x1'] +
                                              xw[1] * db['x2']))
    error_i = error_i - np.mean(error_i)
```

```
    #utilize the weights to account for heteroscedasticity
    db["resid_i"] = np.log(error_i ** 2)
    h_i = np.exp(smf.ols(formula = "resid_i ~ x1 + x2",
                         data = db).fit().fittedvalues)
    weights_i = 1 / h_i

    #simulate the target
    db['y_sim'] = db['y'] + error_i

    #run the regressions
    lr_unw = smf.ols(formula = "y_sim ~ x1 + x2",
                     data = db).fit()
    lr_wts = sm.WLS.from_formula(formula = "y_sim ~ x1 + x2",
                                 weights = weights_i,
                                 data = db).fit()

    res_unw.loc[i, ] = lr_unw.params[1:].tolist() + lr_unw.pvalues[1:].tolist()
    res_wts.loc[i, ] = lr_wts.params[1:].tolist() + lr_wts.pvalues[1:].tolist()

#confidence intervals
#true_betas
{"beta1": 0.85, "beta2": -0.65}
```

```
## {'beta1': 0.85, 'beta2': -0.65}
```

```
#unweighted betas
b12_ci_unw = res_unw[["b1", "b2"]].quantile([alpha / 2, 0.5, 1 - alpha / 2])
b12_ci_unw
```

```
##              b1        b2
## 0.025  0.188296 -1.221166
## 0.500  0.859330 -0.649491
## 0.975  1.494286 -0.061841
```

```
#weighted betas
b12_ci_wts = res_wts[["b1", "b2"]].quantile([alpha / 2, 0.5, 1 - alpha / 2])
b12_ci_wts
```

```
##              b1        b2
## 0.025  0.552355 -1.023943
## 0.500  0.849808 -0.652018
```

```
## 0.975  1.137475 -0.305847
```

```
#power
#heteroscedastic errors
(res_unw[["b1.pval", "b2.pval"]] < alpha).mean()
```

```
## b1.pval    0.862
## b2.pval    0.666
## dtype: float64
```

```
#heteroscedastic errors modeled
(res_wts[["b1.pval", "b2.pval"]] < alpha).mean()
```

```
## b1.pval    1.000
## b2.pval    0.953
## dtype: float64
```

While the above simulation confirms the commonly accepted consequences of heteroscedasticity, readers are encouraged to modify and test different setups and explore various methods of addressing heteroscedastic errors.