ARE212- FINAL

Anaya Hall

May 10, 2018

ARE212 Take-home Final: Exploring Measurement Error

In this assessment we explore what happens to the least squares estimator for varying degrees of measurement error in the left hand and right hand side variables. Assume the following population model:

$y_i^* = \beta_0 + \beta_1 \cdot x_{1i} + \beta_2 \cdot x_{2i} + \epsilon_i$

Assume: $E[\epsilon|x] = 0$ and ϵ is drawn from a normal distribution (0,1)

x_{1i} and x_{2i} drawn from uniform normal [-200, 200]

$\beta_0 = 1$, $\beta_1 =$ -0.75, $\beta_2 = 0.75$

Set seed to 22092008.

1. Generate random sample

```
# Generate population data:
# Set a seed
set.seed(22092008)

# Set the population size
N <- 1e2

# Generate the data for X and E
x1 = runif(n = N, min = -200, max = 200)
x2 = runif(n = N, min = -200, max = 200)
e = rnorm(n = N, mean = 0, sd = 1)

# Generate the y variables (in anticipataion of question #2)
y = 1 -0.75*x1 + 0.75*x2 + e

# Join the data together
pop_df <- as.data.frame(cbind(y, x1, x2))
```

2. Generate y's and estimate b's with OLS

Generate y variables (done above). Estimate the three β coefficients using least squares. Calculate the difference between the true β and the estimated coefficient for each of the three β coefficients.

First, let's load our OLS functions from the last few problem sets:

```
b_ols <- function(y, X) {
  # Calculate beta hat
  beta_hat <- solve(t(X) %*% X) %*% t(X) %*% y
```

```
  # Return beta_hat
  return(beta_hat)
}
```

Now ready to estimate!

```
true_b <- matrix(c(1,-.75,.75), ncol=1)
# Select X matrix, add intercept column and convert to matrix
X_mat <- pop_df[,2:3] %>% cbind(1,.) %>% as.matrix()

# Run OLS
st2 <- b_ols(pop_df$y, X_mat)

# Calculate BIAS
bias_mat_s2 <- data_frame(
  bias_0 = (st2[1] - 1),
  bias_1 = (st2[2] - (-0.75)),
  bias_2 = (st2[3] - (0.75)))

bias_mat_s2 %>% knitr::kable()
```

bias_0	bias_1	bias_2
-0.0925107	0.0001089	-0.0001058

3. Introducing MEASUREMENT ERROR on y

Now assume that y_i is measured with error! It is in fact $y_i + r_i$, where r_i is drawn from a random normal distribution with mean zero and variance σ^2. Using the y_i from the previous step, add the measurement error r_i to them and use this measured with error dependent variable as your outcome.

Estimate the three β coefficients again using least squares. Do this for $\sigma^2 = [1\ 10\ 100]$.

```
# Again select X matrix, add intercept column and convert to matrix
X_mat <- pop_df[,2:3] %>% cbind(1,.) %>% as.matrix()

sigma2 <- c(1,10,100)

bias_mat <- array(NA, dim = c(3,3))

for (s in sigma2) {
  i <- which(sigma2 == s)
  # Generate r_i
  set.seed(22092008)
  r <- rnorm(N, 0, s)
  # Add new error to y
  pop_df %<>% mutate(y_err = y + r)
  # Rerun OLS
  bias_mat[,i] <- (b_ols(pop_df$y_err, X_mat) - true_b)
  row.names(bias_mat) <- c("Intercept", "X1", "X2")
```

```
}

bias_mat %>% knitr::kable(col.names = c("sigma2: 1", "sigma2: 10", "sigma2: 100"))
```

	sigma2: 1	sigma2: 10	sigma2: 100
Intercept	0.1327935	2.1605317	22.4379135
X1	-0.0000617	-0.0015971	-0.0169517
X2	0.0006147	0.0070989	0.0719406

Starting to see some more bias (that is, our coefficients are further from truth than in step 2). The intercept seems to be absorbing a lot of this error, which we can see in the larger bias. The bias increases with increasing variance of r_i.

4. Measurement Error on x2

Now assume that your x_{2i} is measured with error. You observe $x_2^* = x_{2i} + r_i$, where r_i is drawn from a random normal distribution with mean zero and variance σ^2 (Just reuse the r_i from the previous step.). Use the y_i from the first step (the one measured without error) and replace x_{2i} with x_2^* in your estimation. Estimate the three β coefficients using least squares on your data. Do this for $\sigma^2 = [1\ 10\ 100]$.

```
set.seed(22092008)
sigma2 <- c(1,10,100)

bias_mat <- array(NA, dim = c(3,3))

for (s in sigma2) {
  i <- which(sigma2 == s)
  # Generate r_i
  set.seed(22092008)
  r <- rnorm(N, 0, s)
  # Add new error to y
  pop_df %<>% mutate(x2_star = x2 + r)

  X_mat_star <- pop_df[['x1']] %>% cbind(. , pop_df[['x2_star']]) %>% cbind(1,.) %>% as.matrix()

  # Rerun OLS
  bias_mat[,i] <- (b_ols(pop_df$y, X_mat_star) - true_b)
  row.names(bias_mat) <- c("Intercept", "X1", "X2")

}

bias_mat %>% knitr::kable(col.names = c("sigma2: 1", "sigma2: 10", "sigma2: 100"))
```

	sigma2: 1	sigma2: 10	sigma2: 100
Intercept	-0.2621724	-1.8323206	-14.0642622
X1	0.0002353	0.0012482	0.0008541
X2	-0.0007033	-0.0104784	-0.3103493

Now we start to see a good amount of bias on the coefficient on X2 (althuogh the coefficient on the intercept still has the most bias.) Again, the bias increases with increasing variance of r_i.

5. Non-symmetric measurement error: POSITIVE

Repeat step 4 exactly, only now assume that your measurement error is not symmetric, but always positive. Simply take the absolute value of r_i (again using the r_i from above) before generating your x_2^*. Only do this for $\sigma^2 = 100$.

```
bias_mat <- array(NA, dim = c(3,1))

set.seed(22092008)
r <- rnorm(N, 0, 100) %>% abs()
# Add new error to y
pop_df %<>% mutate(x2_star = x2 + r)

X_mat_star <- pop_df[['x1']] %>% cbind(. , pop_df[['x2_star']]) %>% cbind(1,.) %>% as.matrix()

# Rerun OLS
bias_mat[,1] <- (b_ols(pop_df$y, X_mat_star) - true_b)
row.names(bias_mat) <- c("Intercept", "X1", "X2")

bias_mat %>% knitr::kable(col.names = c("sigma2: 100"))
```

	sigma2: 100
Intercept	-52.9488654
X1	0.0498078
X2	-0.1702244

Here, we're just looking at the bias when $\sigma^2 = 100$ and the measurement error is positive. The bias on each β coefficient has ~doubled with this non-symmetric measurement error.

6. Non-symmetric measurement errror: NEGATIVE

Repeat step 4 exactly, only now assume that your measurement error is not symmetric, but always negative. Simply take the absolute value of r_i and multiply it times (-1) before generating your x_2^* (again using the r_i from above). Only do this for $\sigma^2 = 100$.

```
bias_mat <- array(NA, dim = c(3,1))

set.seed(22092008)
r <- rnorm(N, 0, 100) %>% abs()
# Add new error to y
pop_df %<>% mutate(x2_star = x2 + (-r))

X_mat_star <- pop_df[['x1']] %>% cbind(. , pop_df[['x2_star']]) %>% cbind(1,.) %>% as.matrix()

# Rerun OLS
bias_mat[,1] <- (b_ols(pop_df$y, X_mat_star)  - true_b)
```

```r
row.names(bias_mat) <- c("Intercept", "X1", "X2")


bias_mat %>% knitr::kable(col.names = c("sigma2: 100"))
```

	sigma2: 100
Intercept	51.8165984
X1	-0.0605573
X2	-0.1306192

Again, the bias has about doubled from step 4 with non-symmetric measurement error. The coefficients on the intercept as well as x_1 have both also changed sign!

Time to SCALE UP!

Repeat the above steps 10,000 times (set the seed only the first time, not each time) and calculate the bias for β_0, β_1 and β_2 for each setting and fill in the table below.

```r
sim_array <-  array(NA, dim = c(1e5, 3, 3))

one_iter <- function(N) {

  # Generate the data for X and E
  x1 = runif(n = N, min = -200, max = 200)
  x2 = runif(n = N, min = -200, max = 200)
  e = rnorm(n = N, mean = 0, sd = 1)
  # Generate the y variables (in anticipataion of question #2)
  y = 1 -0.75*x1 + 0.75*x2 + e

  # Join the data together
  pop_df <- as.data.frame(cbind(y, x1, x2))
  X_mat <- pop_df[,2:3] %>% cbind(1,.) %>% as.matrix()

  #Step 3: Meas. Error in Y

  sigma2 <- c(1,10,100)

  bias_mat_s3 <- array(NA, dim = c(3,3))

  for (s in sigma2) {
    i <- which(sigma2 == s)
    # Generate r_i
    r <- rnorm(N, 0, s)
    # Add new error to y
    pop_df %<>% mutate(y_err = y + r)
    # Rerun OLS
    bias_mat_s3[,i] <- (b_ols(pop_df$y_err, X_mat) - true_b)
    row.names(bias_mat_s3) <- c("Intercept", "X1", "X2")
```

```
}

#Step 4: Meas. Error in X2

bias_mat_s4 <- array(NA, dim = c(3,3))

for (s in sigma2) {
  i <- which(sigma2 == s)
  # Generate r_i
  # set.seed(seed)
  r <- rnorm(N, 0, s)
  # Add new error to y
  pop_df %<>% mutate(x2_star = x2 + r)

  X_mat_star <- pop_df[['x1']] %>% cbind(. , pop_df[['x2_star']]) %>% cbind(1,.) %>% as.matrix()

  # Rerun OLS
  bias_mat_s4[,i] <- (b_ols(pop_df$y, X_mat_star) - true_b)
  row.names(bias_mat_s4) <- c("Intercept", "X1", "X2")

}

#Step 5: POS. Meas. Error in X2

bias_mat_s5 <- array(NA, dim = c(3,3))

for (s in sigma2) {
  i <- which(sigma2 == s)
  # Generate r_i
  # set.seed(seed)
  r <- rnorm(N, 0, s)
  # Add new error to y
  pop_df %<>% mutate(x2_star = x2 + abs(r))

  X_mat_star <- pop_df[['x1']] %>% cbind(. , pop_df[['x2_star']]) %>% cbind(1,.) %>% as.matrix()

  # Rerun OLS
  bias_mat_s5[,i] <- (b_ols(pop_df$y, X_mat_star) - true_b)
  row.names(bias_mat_s5) <- c("Intercept", "X1", "X2")

}

#Step 6: NEG. Meas. Error in X2

bias_mat_s6 <- array(NA, dim = c(3,3))

for (s in sigma2) {
  i <- which(sigma2 == s)
  # Generate r_i
```

```
    # set.seed(seed)
    r <- rnorm(N, 0, s)
    # Add new error to y
    pop_df %<>% mutate(x2_star = x2 - abs(r))

    X_mat_star <- pop_df[['x1']] %>% cbind(. , pop_df[['x2_star']]) %>% cbind(1,.) %>% as.matrix()

    # Rerun OLS
    bias_mat_s6[,i] <- (b_ols(pop_df$y, X_mat_star) - true_b)
    row.names(bias_mat_s6) <- c("Intercept", "X1", "X2")

  }

  results_df <- rbind(bias_mat_s3, bias_mat_s4, bias_mat_s5, bias_mat_s6)
  colnames(results_df) <- c("sigma_1", "sigma_10", "sigma_100")

  return(results_df)

}

# test to make sure it works!
one_iter(100)
```

```
##                 sigma_1     sigma_10    sigma_100
## Intercept  0.2804519228  0.429865554  -4.42847955
## X1        -0.0016670073 -0.001337710  -0.15386786
## X2         0.0007866190  0.007408139   0.12352867
## Intercept  0.1143776537  2.130742344   2.26281527
## X1        -0.0004724873 -0.006554161   0.03980157
## X2         0.0019688076 -0.005240375  -0.38616454
## Intercept -0.4749545906 -6.177642025 -43.74522514
## X1        -0.0001911570  0.003349814   0.04454457
## X2         0.0008762871 -0.010022576  -0.19361494
## Intercept  0.7877347418  6.595116193  46.36582709
## X1         0.0001854594 -0.002214805  -0.00230751
## X2         0.0005951370 -0.003825216  -0.15815194
```

```
n_iter <- 10000
seed <- 22092008

# Prepare 3-d array for storing results
big_bias_mat <- array(NA, dim = c(n_iter, 12,3))

set.seed(seed)

# Run 10000 simulations
for (i in 1:n_iter) {
  big_bias_mat[i,,] <- one_iter(100)
}

# Create array for average of results
```

```
mean_bias <- array(NA, c(12,3))
colnames(mean_bias) <- c("sigma_1", "sigma_10", "sigma_100")
row.names(mean_bias) <- c("3Intercept", "3X1", "3X2","4Intercept", "4X1", "4X2","5Intercept", "5X1

# Loop to take mean across simulations and save output
for (i in 1:12) {
    for (j in 1:3) {
        mean_bias[i,j] <- mean(big_bias_mat[,i,j])
  } }

bias_mat_s2         # Original bias results from step 2
```

```
## # A tibble: 1 x 3
##    bias_0   bias_1    bias_2
##     <dbl>    <dbl>     <dbl>
## 1 -0.0925 0.000109 -0.000106
```

```
t(mean_bias)        # Transpose results so easier to read into table below!
```

```
##             3Intercept           3X1          3X2    4Intercept
## sigma_1   -0.001013074 -7.139338e-06 6.620332e-06 -0.0001427097
## sigma_10   0.002766648 -1.273098e-05 1.046166e-04 -0.0108522832
## sigma_100  0.025350622  6.636537e-04 8.169639e-05  0.0927713100
##                    4X1           4X2  5Intercept           5X1
## sigma_1   -2.201842e-05 -4.033294e-05  -0.5998853 -5.923982e-06
## sigma_10   1.101392e-05 -5.479777e-03  -5.9642780 -7.573438e-05
## sigma_100  1.483872e-03 -3.205047e-01 -47.0630331  5.221453e-04
##                    5X2 6Intercept           6X1           6X2
## sigma_1   -1.540449e-06  0.5968624 -4.798518e-06 -8.917522e-06
## sigma_10  -2.020024e-03  5.9696288  7.254783e-05 -1.990584e-03
## sigma_100 -1.594324e-01 47.1203795  7.107956e-04 -1.587771e-01
```

Final results table

	Bias β_0	Bias β_1	Bias β_2
Step 2	-0.0925107	0.0001089	-0.0001058
Step 3 ($\sigma^2 = 1$)	-0.001013074	-7.139338e-06	6.620332e-06
Step 3 ($\sigma^2 = 10$)	0.002766648	-1.273098e-05	1.046166e-04
Step 3 ($\sigma^2 = 100$)	22.4379135	6.636537e-04	8.169639e-05
Step 4 ($\sigma^2 = 1$)	-0.0001427097	-2.201842e-05	-4.033294e-05
Step 4 ($\sigma^2 = 10$)	-0.0108522832	1.101392e-05	-5.479777e-03
Step 4 ($\sigma^2 = 100$)	0.0927713100	1.483872e-03	-3.205047e-01
Step 5 ($\sigma^2 = 100$)	-47.0630331	5.221453e-04	-1.594324e-01
Step 6 ($\sigma^2 = 100$)	47.1203795	7.107956e-04	-1.587771e-01

Note- I calculated bias as the estimated coefficient less the true coefficient

It seems like measurement error is biasing our estimates towards zero for the most part. The bias is worse (that is, bigger!) with the measurement error has larger variance, which is what I would have expected. As

described earlier, it also seems to have the largest impact on the coefficient on the intercept. I am surprised that the bias is similar with positive and negative non-symmetric measurement error (steps 5 and 6).

Unrelated to bias or measurement error- I learned how to deal with slash think about three-dimensional arrays on this assessment. I consider that a win!

P.S. I'm not sure I set up the simulation correctly, but I did get it to run (unlike in the take-home midterm when the simulation crashed my computer). Thank you for a lovely semester and making econometrics enjoyable :)