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

```r
# 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:

```r
# Function to convert tibble, data.frame, or tbl_df to matrix
to_matrix <- function(the_df, vars) {
  # Create a matrix from variables in var
```

```
  new_mat <- the_df %>%
    #Select the columns given in 'vars'
    select_(.dots = vars) %>%
    # Convert to matrix
    as.matrix()
  # Return 'new_mat'
  return(new_mat)
}


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

ols <- function(data, y_data, X_data, intercept = T, hetsked = F, H0 = 0, two_tail = T, alpha = 0
  # Function setup ----
    # Require the 'dplyr' package
    require(dplyr)

  # Create dependent and independent variable matrices ----
    # y matrix
    y <- to_matrix (the_df = data, vars = y_data)
    # X matrix
    X <- to_matrix (the_df = data, vars = X_data)
      # If 'intercept' is TRUE, then add a column of ones
      if (intercept == T) {
      X <- cbind(1,X)
      colnames(X) <- c("intercept", X_data)
      }

  # Calculate b, y_hat, and residuals ----
    b <- solve(t(X) %*% X) %*% t(X) %*% y
    y_hat <- X %*% b
    e <- y - y_hat

    # Inverse of X'X
    XX <- t(X) %*% X
    XX_inv <- solve(t(X) %*% X)

    if (hetsked == T) {
      # For each row, calculate x_i' x_i e_i^2; then sum
     sigma_hat <- lapply(X = 1:n, FUN = function(i) {
      # Define x_i
      x_i <- matrix(as.vector(X[i,]), nrow = 1)
      # Return x_i' x_i e_i^2
      return(t(x_i) %*% x_i * e[i]^2)
      }) %>% Reduce(f = "+", x = .) }
```

```r
  if (hetsked == F) sigma_hat <- XX

# Useful -----
  n <- nrow(X) # number of observations
  k <- ncol(X) # number of independent variables
  dof <- n - k # degrees of freedom
  i <- rep(1,n) # column of ones for demeaning matrix
  A <- diag(i) - (1 / n) * i %*% t(i) # demeaning matrix
  y_star <- A %*% y # for SST
  X_star <- A %*% X # for SSM
  SST <- drop(t(y_star) %*% y_star)
  SSM <- drop(t(b) %*% t(X_star) %*% X_star %*% b)
  SSR <- drop(t(e) %*% e)

# Measures of fit and estimated variance ----
  R2uc <- drop((t(y_hat) %*% y_hat)/(t(y) %*% y)) # Uncentered R^2
  R2 <- 1 - SSR/SST # Uncentered R^2
  R2adj <- 1 - (n-1)/dof * (1 - R2) # Adjusted R^2
  AIC <- log(SSR/n) + 2*k/n # AIC
  SIC <- log(SSR/n) + k/n*log(n) # SIC
  s2 <- SSR/dof # s^2

# Measures of fit table ----
  mof_table_df <- data.frame(R2uc, R2, R2adj, SIC, AIC, SSR, s2)
  mof_table_col_names <- c("$R^2_\\text{uc}$", "$R^2$",
                           "$R^2_\\text{adj}$",
                           "SIC", "AIC", "SSR", "$s^2$")
  mof_table <-  mof_table_df %>% knitr::kable(
    row.names = F,
    col.names = mof_table_col_names,
    format.args = list(scientific = F, digits = 4),
    booktabs = T,
    escape = F
  )

# t-test----
  # Standard error
  se <- sqrt(s2 * diag(XX_inv %*% sigma_hat %*% XX_inv)) # Vector of _t_ statistics
  # Vector of _t_ statistics
  t_stats <- (b - H0) / se
  # Calculate the p-values
  if (two_tail == T) {
  p_values <- pt(q = abs(t_stats), df = dof, lower.tail = F) * 2
  } else {
    p_values <- pt(q = abs(t_stats), df = dof, lower.tail = F)
  }
  # Do we (fail to) reject?
  reject <- ifelse(p_values < alpha, reject <- "Reject", reject <- "Fail to Reject")
```

```r
    # Nice table (data.frame) of results
    ttest_df <- data.frame(
      # The rows have the coef. names
      effect = rownames(b),
      # Estimated coefficients
      coef = as.vector(b) %>% round(3),
      # Standard errors
      std_error = as.vector(se) %>% round(4),
      # t statistics
      t_stat = as.vector(t_stats) %>% round(3),
      # p-values
      p_value = as.vector(p_values) %>% round(4),
      # reject null?
      significance = as.character(reject)
      )

    ttest_table <-  ttest_df %>% knitr::kable(
      col.names = c("", "Coef.", "S.E.", "t Stat", "p-Value", "Decision"),
      booktabs = T,
      format.args = list(scientific = F),
      escape = F,
      caption = "OLS Results"
    )

  # Data frame for exporting for y, y_hat, X, and e vectors ----
    export_df <- data.frame(y, y_hat, e, X) %>% tbl_df()
    colnames(export_df) <- c("y","y_hat","e",colnames(X))

  # Return ----
    return(list(n=n, dof=dof, b=b, se=se, vars=export_df, R2uc=R2uc,R2=R2,
                R2adj=R2adj, AIC=AIC, SIC=SIC, s2=s2, SST=SST, SSR=SSR,
                mof_table=mof_table, ttest=ttest_table))
}
```

Now ready to estimate!

```r
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 <- data_frame(
  bias_0 = (st2[1] - 1),
  bias_1 = (st2[2] - (-0.75)),
  bias_2 = (st2[3] - (0.75)))

bias_mat %>% 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 $\sigma^2 = [1\ 10\ 100]$.

```r
# 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("sigma^2: 1", "sigma2: 10", "sigma2: 100"))
```

	sigma^2: 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 $\sigma^2 = [1\ 10\ 100]$.

```r
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	
Intercept	-0.2621724	-1.8323206	
X1	0.0002353	0.0012482	
X2	-0.0007033	-0.0104784	
Now we start	to see a goo	d amount of b	ias on the coefficient on X2 (althuogh the coefficient on the intercept s

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 $\sigma^2 = 100$.

```r
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"))
```

Intercept
X1
X2
Here, we're just looking at the bias when $\sigma^2 = 100$ and the measurement error is positive. The bias on each β coef

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


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

Intercept
X1
X2
Again, the bias has about doubled from step 4 with non-symmetric measurement error. The coefficients on the int

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.

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



one_iter <- function(N) {

  set.seed(seed)
```

```r
# 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()

sigma2 <- c(1,10,100)

#Step 3: Meas. Error in Y

bias_mat_s3 <- 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(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")
```

```r
  }

  #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)

  return(results_df)

}

#one_iter(100)
```

```
n_iter <- 10
seed <- 22092008

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

set.seed(seed)

for (i in 1:n_iter) {

  big_bias_mat[i,,] <- one_iter(100)

}

results <- array(12,3)

# for (i in 1:12) & (j in 1:3) {
#         print(i,j)
#         #results[i,j] <- mean(big_bias_mat[,j,i])
#     }


# bias_sim <- function(n_sims, sample_size, seed = 12345) {
#   # Set the seed
#   set.seed(seed)
#   # Run one_sim n_sims times; convert results to data.frame
#   sim_df <- replicate(
#     n = n_sims,
#     expr = one_iter(sample_size),
#     simplify = F
#     )
#   # Return sim_df
#   return(sim_df)
# }
#
# bias_sim(50, 100, seed =22092008)
```

What have you learned from this exercise that you did not know before? Was there anything surprising?

	Bias β_0	Bias β_1	Bias β_2
Step 2	-0.0925107	0.0001089	-0.0001058
Step 3 ($\sigma^2 = 1$)	0.1327935	-6.169714e-05	6.147005e-04
Step 3 ($\sigma^2 = 10$)	2.1605317	-1.597147e-03	0.0070989
Step 3 ($\sigma^2 = 100$)	22.4379135	-1.695165e-02	0.071940
Step 4 ($\sigma^2 = 1$)	-2.293300e-02	2.755397e-05	-1.302244e-05
Step 4 ($\sigma^2 = 10$)	-1.3014183389	-2.221322e-03	4.130867e-03
Step 4 ($\sigma^2 = 100$)	-10.111527	-2.499077e-02	-3.085128e-01
Step 5 ($\sigma^2 = 100$)	-49.7100828	0.0675275515	-0.1374962359
Step 6 ($\sigma^2 = 100$)	48.96150816	0.02880406	-0.14629597