Pol Sci 630: Problem Set 10: Functional Form, Endogeneity, Power

Prepared by: Anh Le (anh.le@duke.edu)

Due Date: Nov 9 (Beginning of Class)

1 Functional specification

There's a famous dataset called the Anscombe quartet. You load it in R like so:

```
head(anscombe)
```

```
##   x1 x2 x3 x4   y1   y2    y3   y4
## 1 10 10 10  8 8.04 9.14  7.46 6.58
## 2  8  8  8  8 6.95 8.14  6.77 5.76
## 3 13 13 13  8 7.58 8.74 12.74 7.71
## 4  9  9  9  8 8.81 8.77  7.11 8.84
## 5 11 11 11  8 8.33 9.26  7.81 8.47
## 6 14 14 14  8 9.96 8.10  8.84 7.04
```

1.1 Explore Anscombe

There are 4 pairs of x and y. Run 4 regressions of y on x. Check out the regression result. A bit late for Halloween, but what spooky thing do you notice?

Then plot the data.

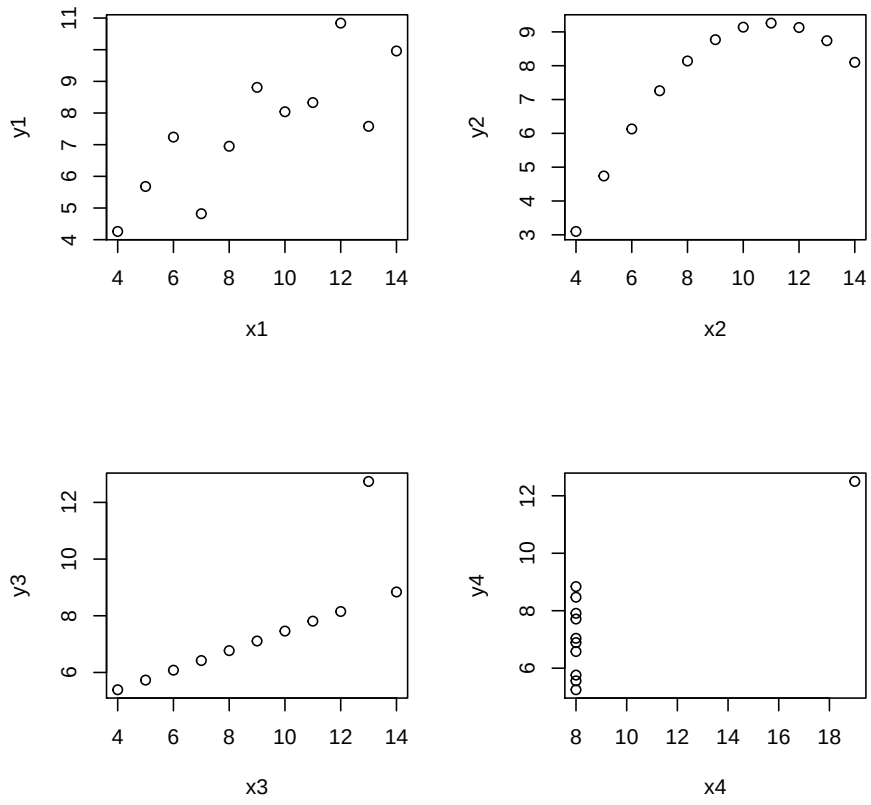
Solution

```
(m1 <- lm(y1 ~ x1, data = anscombe))
```

```
##
## Call:
## lm(formula = y1 ~ x1, data = anscombe)
##
## Coefficients:
## (Intercept)           x1
##      3.0001       0.5001
```

```
(m2 <- lm(y2 ~ x2, data = anscombe))
```

```
##
## Call:
## lm(formula = y2 ~ x2, data = anscombe)
##
## Coefficients:
## (Intercept)           x2
##       3.001        0.500

(m3 <- lm(y3 ~ x3, data = anscombe))

##
## Call:
## lm(formula = y3 ~ x3, data = anscombe)
##
## Coefficients:
## (Intercept)           x3
##      3.0025       0.4997

(m4 <- lm(y4 ~ x4, data = anscombe))

##
## Call:
## lm(formula = y4 ~ x4, data = anscombe)
##
## Coefficients:
## (Intercept)           x4
##      3.0017       0.4999

par(mfrow=c(2,2))
plot(y1 ~ x1, data = anscombe)
plot(y2 ~ x2, data = anscombe)
plot(y3 ~ x3, data = anscombe)
plot(y4 ~ x4, data = anscombe)
```




```
par(mfrow=c(1,1))
```

The four regression models have exactly the same coefficients, standard errors, p-value, R-square (!). But the pattern of the data looks very different. A good lesson about why we visualize data and don't run regression blindly.

1.2 Ramsey RESET

Use Ramsey RESET on the 4 models to check. Which kind of functional misclassification can it catch?

Solution

```
library(lmtest)

## Loading required package:  zoo
```

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

```
resettest(m1, power = 2, type = "fitted")
```

```
##
##  RESET test
##
## data:  m1
## RESET = 0.5318, df1 = 1, df2 = 8, p-value = 0.4866
```

```
resettest(m2, power = 2, type = "fitted")
```

```
##
##  RESET test
##
## data:  m2
## RESET = 4925000, df1 = 1, df2 = 8, p-value < 2.2e-16
```

```
resettest(m3, power = 2, type = "fitted")
```

```
##
##  RESET test
##
## data:  m3
## RESET = 0.46605, df1 = 1, df2 = 8, p-value = 0.5141
```

```
resettest(m4, power = 2, type = "fitted")
```

```
##
##  RESET test
##
## data:  m4
## RESET = 0, df1 = 1, df2 = 8, p-value = 1
```

The RESET test can only catch the second case, where there is a curvillinear relationship. This is because RESET simply adds squared terms and check whether they are important. Even though it's problematic to say the 3rd and 4th models are a good fit, RESET can't catch it.

This is to show what a test can and cannot do.

Table 1: Signing the bias

	$\beta_2 > 0$	$\beta_2 < 0$
$\delta_1 > 0$		
$\delta_1 < 0$		

2 Endogeneity – Omitted Variable Bias

2.1 Sign the bias – math

Given the following data generating process (DGP)

$$x_2 = \delta_0 + \delta_1 x_1 + v \tag{1}$$
$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + w \tag{2}$$

The following equation (lecture 09/26) shows what happens when we regress y on x_1, omitting x_2. **Prove this equation**.

$$y = (\beta_0 + \beta_2\delta_0) + (\beta_1 + \beta_2\delta_1)x_1 + (\beta_2 v + w)$$

Solution

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + w \tag{3}$$
$$= \beta_0 + \beta_1 x_1 + \beta_2(\delta_0 + \delta_1 x_1 + v) + w \tag{4}$$
$$= (\beta_0 + \beta_2\delta_0) + (\beta_1 + \beta_2\delta_1)x_1 + (\beta_2 v + w) \tag{5}$$

2.2 Sign the bias - simulation

In the equation you proved above, the estimated coefficient for x_1 is $\beta_1 + \beta_2\delta_1$, different from its true value β_1. The bias is $\beta_2\delta_1$. The sign of the bias thus depends on β_2 and δ_1, as discussed in the lecture and reproduced in Table 1.

Conduct 4 simulations with appropriate values of β_2 and δ_1 corresponding to the 4 cells in the table. Show that the sign of the bias is as we learned in class.

Solution

We create a function that calculates the bias depending on the values of β_2 and δ_1.

```
delta1 <- 1 ; beta2 <- 3 # These 2 are keys

bias_simulation <- function(beta2, delta1) {
  beta0 <- 1 ; beta1 <- 2 ; delta0 <- 1 # These don't matters for bias sign
  x2 <- rnorm(100)
  x1 <- delta1 * x2 + rnorm(100)
```

```
  y <- beta0 + beta1 * x1 + beta2 * x2 + rnorm(100)

  estimated_beta1 <- coef(lm(y ~ x1))["x1"]
  bias <- estimated_beta1 - beta1
  return(bias)
}
```

We use this function to show the sign of the bias in each of the 4 cells

```
bias_simulation(beta2 = 1, delta1 = 1)

##        x1
## 0.5484303

bias_simulation(beta2 = 1, delta1 = -1)

##         x1
## -0.5060759

bias_simulation(beta2 = -1, delta1 = -1)

##        x1
## 0.2718827

bias_simulation(beta2 = -1, delta1 = 1)

##       x1
## -0.66955
```

We confirm that whenever β_2 and δ_1 are of opposite sign, we have negative bias.

3 Power calculation

In this exercise we practice power calculation for the simplest experiment setup.

Assume that our binary treatment has an effect size of 2 on the outcome, as follows:

$$y = 1 + 2 \times \text{Treatment} + u$$
$$u \sim Normal(mean = 1, sd = 10)$$

In our experiment, we randomly assigned n experimental units into 2 groups, treated and control, i.e. treatment $= 1$ and treatment $= 0$. Calculate the power of our experiment (i.e. the probability that we can reject the null of zero treatment effect) for different values of the sample size n.

The end product I want to see is a graph with n on the x-axis and power on the y-axis. How big must your sample size be to get a power of 0.8?

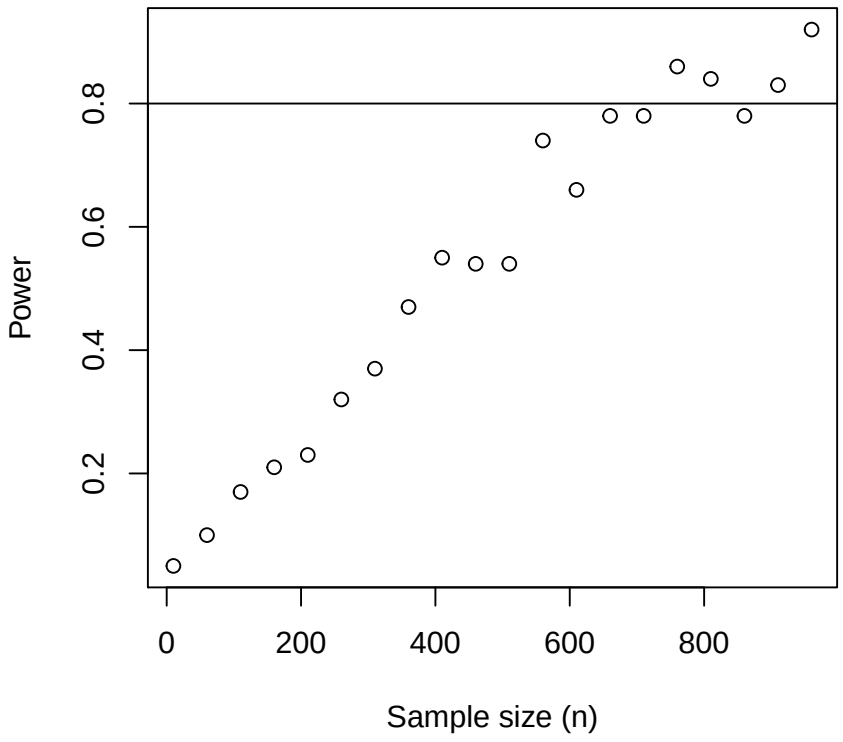
Solution

We first create a function that takes a value of n and calculate the power.

```
power_sim <- function(n) {
  number_of_simulations <- 100
  pvalues <- rep(NA, number_of_simulations)
  for (i in 1:number_of_simulations) {
    treatment <- sample(c(0, 1), size = n, replace = TRUE)
    y <- 1 + 2 * treatment + rnorm(n, mean = 1, sd = 10)
    pvalues[i] <- coef(summary(lm(y ~ treatment)))[2, 4]
  }
  mean(pvalues < 0.05)
}
```

We then apply this function to a range of n values to get the corresponding power.

```
set.seed(1)
ns <- seq(from = 10, to = 1000, by = 50)
powers <- sapply(ns, power_sim)
plot(powers ~ ns, ylab = "Power", xlab = "Sample size (n)")
abline(h = 0.8)
```



The plot shows that we need about 700-800 units to reach the commonly accepted power level of 0.8.