Imbens and Rubin Chapter 3

Alex Stephenson

Notation

We have a population of N units, where each unit in the population has a set of pre-treatment covariates. In this section, Imbens and Rubin are setting up a matrix. Here is an example of what they mean with a constant treatment effect of 1.

```
set.seed(42)
N <- 100 # number of units in the population
x1 <- rnorm(100) # pretreatment covariate 1
x2 <- rnorm(100) # pretreatment covariate 2
x3 <- rnorm(100) # pretreatmetn covariate 3
Yi0 <- runif(100) # potential outcome under control
Yi1 <- Yi0 + 1 # potential outcome under treatment

# column binding
covariate_matrix <- cbind(N, Yi0, Yi1, x1, x2, x3)
covariate_matrix[1:5,]
```

```
##        N       Yi0      Yi1         x1         x2         x3
## [1,] 100 0.4981566 1.498157  1.3709584  1.2009654 -2.0009292
## [2,] 100 0.2827642 1.282764 -0.5646982  1.0447511  0.3337772
## [3,] 100 0.7764451 1.776445  0.3631284 -1.0032086  1.1713251
## [4,] 100 0.3038528 1.303853  0.6328626  1.8484819  2.0595392
## [5,] 100 0.5155512 1.515551  0.4042683 -0.6667734 -1.3768616
```

We will see representations like this a lot in applied work. Now, we need a column vector for treatment assignment. Imbens and Rubin denote treatment assignment by W. For each unit W can be either 0 or 1, where 0 means control and 1 means treated. The authors also provide a way to sum up the number of units in treatment (N_t) and in control (N_c).

```
W <- sample(c(rep(0,50), rep(1,50)), 100, replace = F)

# Add our Treatment assignment to our data matrix
cov_treat_matrix <- cbind(N, Yi0, Yi1, W, x1, x2, x3)
```

The fundamental problem of causal inference is that we can only observe one outcome. This is Equation 3.1 in Imbens and Rubin. The outcomes we observe are the outcomes based on treatment assignment.

```
Y_obs <- ifelse(W == 1, Yi1, Yi0)

all_data <- cbind(N, Yi0, Yi1, W, Y_obs, x1, x2, x3)
observed_dataset <- cbind(N, W, Y_obs, x1, x2, x3)
```

If we conducted an experiment, the result would be the data contained in `observed_dataset`. We print the first four rows for reference.

```
knitr::kable(head(observed_dataset,n = 4))
```

N	W	Y_obs	x1	x2	x3
100	1	1.4981566	1.3709584	1.200965	-2.0009292
100	1	1.2827642	-0.5646982	1.044751	0.3337772
100	0	0.7764451	0.3631284	-1.003209	1.1713251
100	0	0.3038528	0.6328626	1.848482	2.0595392

Assignment Probabilities

The authors use the phrase row exchangeable in the definition of the assignment mechanism. They note this mean that we can change the rows in our dataset without changing the meaning of the assignment mechanism function.

```
# Add a row number to our dataset so we can see what row exchangeability does
row_number <- 1:100
observed_dataset <- cbind(row_number, observed_dataset)
```

If we reorder the rows of the data to be in reverse order[1], this does not change any of the underlying quantities. What will change is what rows are at the top.

```
# Reverse the order of the dataset
knitr::kable(head(observed_dataset[100:1,],4))
```

row_number	N	W	Y_obs	x1	x2	x3
100	100	1	1.4559785	0.6532043	0.1288214	-1.6256167
99	100	0	0.9120300	0.0799826	1.8152284	0.0973405
98	100	1	1.9454572	-1.4592140	0.5864875	0.8625634
97	100	0	0.5347911	-1.1317387	0.4037749	-0.1662615

There is some set notation in this paragraph. $W = 0, 1^N$ is the set of all N-vectors with all elements equal to 0 or 1. To ease computation, imagine we only have 2 units[2]. Here are all the treatment assignment possibilities we could have.

```
W_set <- expand.grid(treat = c(0:1), unit = c(0,1))
W_set
```

```
##   treat unit
## 1     0    0
## 2     1    0
## 3     0    1
## 4     1    1
```

It seems rather strange to put every unit in treatment or control (Assignments 1 and 4), so the Assignment Mechanism function could be made to put 0 probability on those assignment. We might further want to make sure that the second and third assignments are equally likely, so the Assignment Mechanism function could be made to put a probability of 1/2 on each. Such a function corresponds to Example 2 in the chapter (Equation 3.5).

[1] More generally, this holds for any ordering system we choose.
[2] The idea scales to arbitrarily large N, but space on our computer screens do not.