Economics 691-06: Assignment 3

Nihar Shah

Solutions are marked in blue.

Problem 1: Power

(25 points) You need to calculate power for a standard experiment (e.g. no issues regarding noncompliance), and assume you are operating under the following assumptions:

- The ratio of the effect size you want to detect to the standard deviation of noise, i.e. β/σ, is equal to 0.1.
- You are going to reject tests at the 1% level.
- You will always have at least 1000 control units.
- You will never have more than 3000 control units.
- You will never have more than 5000 treatment units.

Plot, on a graph with the x-axis and y-axis as the number of treated and control units respectively, the curve trading off the number of treated and control units you need to get power of 60%; and then add the same curves for 70%, 80%, and 90% power.

We begin by deriving the core equation that gives us power, via standard error of β.

$$\text{SE} = \sigma\sqrt{\frac{1}{n_T} + \frac{1}{n_C}} \quad \text{and} \quad \text{SE} = \frac{\beta}{c_\alpha - \Phi^{-1}(1-\gamma)}$$

$$\sqrt{\frac{1}{n_T} + \frac{1}{n_C}} = \frac{\beta}{\sigma}\frac{1}{c_\alpha - \Phi^{-1}(1-\gamma)}$$

We then plug in the parameters above, including a 1% rejection and $\beta/\sigma = 0.1$.

$$\sqrt{\frac{1}{n_T} + \frac{1}{n_C}} = \frac{0.1}{\Phi^{-1}(0.995) - \Phi^{-1}(1-\gamma)}$$

$$\frac{1}{n_T} + \frac{1}{n_C} = \left(\frac{0.1}{\Phi^{-1}(0.995) - \Phi^{-1}(1-\gamma)}\right)^2$$

At this point, we are largely done. From this equation, if we set two parameters (say, γ and n_C), we can solve out for the third parameter (n_T). The code below largely implements this, fixing two parameters and solving out for the missing one.

However, there is one nuance that emerges, particularly for the high power levels (e.g. $\gamma = 0.9$). No matter how large our treated group is, we have some minimum control group size that is needed;

or vice-versa. For instance, even if we had an infinite number of treated units, we may still need a control group in excess of 1000 units to attain that power. (In computational terms, if we were to fix two parameters, we would be unable to solve for the third.) To find the actual lower bound, which will be useful in the code below, we simply do precisely that trick: we let $n_T \to \infty$, and see what the minimum number of control units would be.

$$\lim_{n_T \to \infty} \frac{1}{n_T} + \frac{1}{n_C} = \left(\frac{0.1}{\Phi^{-1}(0.995) - \Phi^{-1}(1-\gamma)} \right)^2 \quad \Rightarrow \quad n_C = \left(\frac{0.1}{\Phi^{-1}(0.995) - \Phi^{-1}(1-\gamma)} \right)^{-2}$$

Using these equations, we can implement the code to generate the graph.

```
# Load libraries
from scipy.stats import norm
from scipy import optimize
import math
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# Define a function to compute one of two concepts:
# Either compute the residual from the equation, such that we can find the
# pair of (nt, nc) that minimize it
# Or (if flag = False) compute the minimum number of units in each group
def power(nt, nc, power_level, flag = True):
  term = (0.1/(norm.ppf(0.995) - norm.ppf(1 - power_level)))**2
  if flag:
    return(1/nt + 1/nc - term)
  return(1/term)

# Define a function to get the pairs of (nt, nc) for a given power level
def get_pairs(power_level):

  control = []
  treatment = []

  # Find the minimum number of units in the control group
  start = math.ceil(power(0, 0, power_level, False))

  # If this is smaller than 1000, start with 1000 (as we always have that many
  # control units available)
  start = max(start, 1000)

  # Iterate from this starting point up to 3000 (the max no. of control units)
  for nc in range(start, 3001):
    control.append(nc)

    # Define a secondary function that, for a given number of control units and
    # the power level, gets the residual for the number of treated units
```

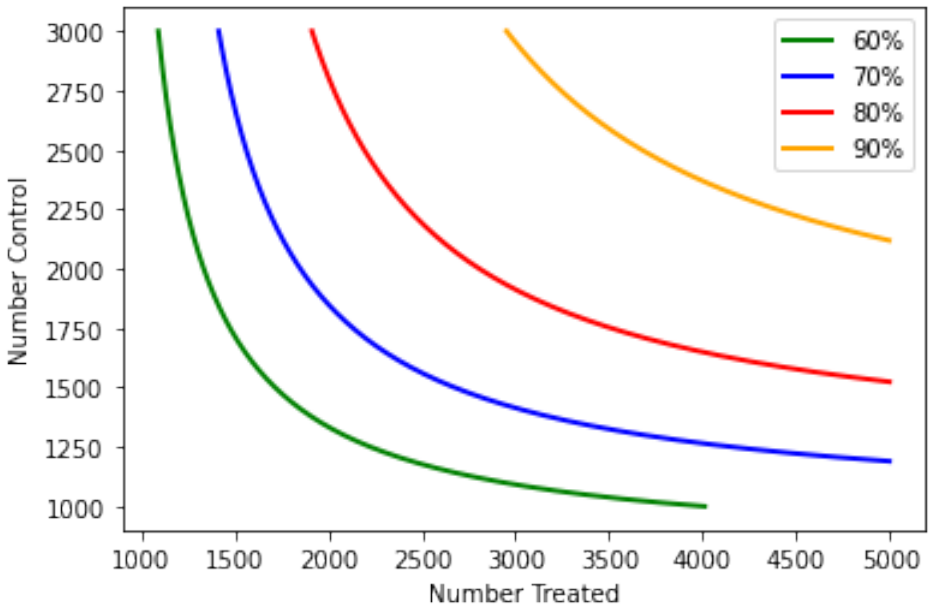
```
    def power_narrow(nt):
      return(power(nt, nc, power_level, True))

    # Find the number of treated units by solving to minimize the residual
    treatment.append(optimize.fsolve(power_narrow, 1)[0])

  # Return data
  output = pd.DataFrame()
  output['nt'] = treatment
  output['nc'] = control

  # If we exceed 5000 treated units, cut those rows; as we only have 5000
  # treated units available
  output = output.iloc[np.where(output['nt'] <= 5000)[0],]
  return(output)

# Generate plots
plt.plot('nt', 'nc', data = get_pairs(0.6), marker = '', color = 'green',
         linewidth = 2, label = "60%")
plt.plot('nt', 'nc', data = get_pairs(0.7), marker = '', color = 'blue',
         linewidth = 2, label = "70%")
plt.plot('nt', 'nc', data = get_pairs(0.8), marker = '', color = 'red',
         linewidth = 2, label = "80%")
plt.plot('nt', 'nc', data = get_pairs(0.9), marker = '', color = 'orange',
         linewidth = 2, label = "90%")
plt.legend()
plt.xlabel("Number Treated")
plt.ylabel("Number Control")
```



Problem 2: Surrogates

Consider the following scenario: Airbnb wants to understand the effect of boosting host earnings in the US for October 1 (for instance, by giving the service fees it collects to hosts) on the number of bookings that a host accepts over the month of October. Unable to wait until the end of October to assess the one-day experiment, Airbnb considers utilizing the surrogacy methodology.

Now consider five variants of this experiment. For each variant, which one of the key assumptions is most likely violated, and why? Limit your answer to one to two sentences per answer.

1. **(5 points)** Airbnb uses the number of bookings that a host accepts on October 1 as the surrogate variable.
   This violates the Surrogacy assumption, as there are other channels by which the treatment can affect the outcome without going through the surrogate. For instance, hosts can add more properties to the site on October 1 in response to the treatment, which culminates in increased bookings later in the month.

2. **(5 points)** Airbnb uses the same surrogate variable, but changes the treatment to run from October 1 through October 3.
   This violates the Surrogacy assumption, but now more directly. Even if the surrogate variable is perfect in theory, the treatment runs for three days while the surrogate variable is only measured on the first day – and so the surrogate definitely fails to fully intermediate the effect of the treatment on the outcome.

3. **(5 points)** Airbnb uses two surrogate variables: the number of bookings that a host accepts from October 1 through October 3, and the number of properties that a host lists from October 1 through October 3. The treatment goes back to being deployed on October 1 only.
   This is deliberately tricky: it is actually not problematic that the surrogate is measured for a longer period than the treatment, and in fact that might be beneficial. Any answer that discusses an additional pathway linking treatment to the outcome, and thus breaking the Surrogacy assumption, would be accepted. For instance, one could argue that hosts take several days to negotiate with and approve guests, and so treatment may affect the surrogate variables at lags longer than three days.

4. **(5 points)** Airbnb uses a long-running observational study from Japan to link its chosen surrogates to the downstream outcome.
   This violates the Comparability assumption, as the link between the surrogates and the outcome in Japan is likely different than the one in the US.

5. **(5 points)** Airbnb uses observational data from the month of August (which is already available and without lags) to identify the treatment effect.
   This violates the Unconfoudedness assumption (i.e. selection bias), as the observational data does not include random variable.

Problem 3: Network Effects

Suppose you are a data scientist at Zynga, working on a new mobile game. Until now, the game has been a single-player experience; but your developers have created a multi-player experience with people in your city. They want to see how much revenue they earn from players with the new multi-player version. Your current single-player game is used in thirty-six cities around the country, and for simplicity, we assume that each city has 1,000 users exactly. You naturally suspect the presence of network effects within any given city.

There are three versions of the experiment you can run. In one version (person-level randomization), you randomize 50% of people in the country (the multi-player version) while keeping 50% in control (the single-player version). In a second version (city-level randomization), you randomize entire cities: all 1,000 users in each city are all given treatment, or all 1,000 users are kept on control. In a third version (two-stage randomization), you first randomize cities to treatment fractions, e.g. you put 60% of Atlanta into treatment, 10% of Austin into treatment, etc; and then you randomize people in each city accordingly. Note that in all cases, the sizes of the treatment and control group are predetermined, e.g. in the first version the experimenter will always have exactly 18,000 units in each group.

The data following *all three* of these experiments is in the file data_assignment3.csv. The file has 36,000 rows. Each row corresponds to a separate user, for which there are seven columns. The first column is the user's city. The second, third, and fourth columns are the their treatment assignment under each of these schematics respectively: treatment_person refers to treatment assignment under person-level randomization, treatment_city refers to treatment assignment under city-level randomization, and treatment_twostage refers to treatment assignment under the two-stage randomization. The fifth, sixth, and seventh columns refer to the outcome (revenue) under each of these randomization schematics.[1]

```
# Load libraries: numpy, pandas, and linear regression
import numpy as np
import pandas as pd
import statsmodels.formula.api as sm

# Load in data
from google.colab import files
import io
uploaded = files.upload()
data = pd.read_csv(io.BytesIO(uploaded['data_assignment3.csv']))
```

1. **(5 points)** Using the first version of the experiment – person-level randomization – estimate the treatment effect and the standard errors.
   To assess the treatment effect in this experiment, we simply compare the average revenue across people in the treated and control group, where treatment and control are defined based on the person-level treatment indicator. Bootstrapping is done by bootstrapping among the

[1] In practice, you will never have the luxury of running three different experimental versions with the same population. However, this design allows you to see the tradeoffs between each version. In fact, the noise introduced for each person is exactly the same under each version, but you will see that the different versions have very different abilities to extract the parameters from the noise!

treated and control groups separately, as the total sizes of the two groups are held constant. Note that there is actually a slightly better way to do this — which is to use linear regression, with the city of each person as an additional covariate — but I do the simpler way to be consistent with later parts of the problem set.

```
# Set seed and identify treated and control observations
np.random.seed(0)
bootstrap = []
treatment = np.where(data['treatment_person'] == 1)[0]
control = np.where(data['treatment_person'] == 0)[0]
bootstrap = []

# Compute the difference in means for the sample (as the estimate)
# And then bootstrap, preserving an exact 50-50 split between people
for i in range(1001):

  if i == 0:
    estimate = (data.iloc[treatment,]['revenue_person'].mean() -
                data.iloc[control,]['revenue_person'].mean())
    print("The estimate is: " + str(np.round(estimate, 2)))

  else:
    treatment_boot = np.random.choice(treatment, len(treatment), replace = True)
    control_boot = np.random.choice(control, len(control), replace = True)
    estimate = (data.iloc[treatment_boot,]['revenue_person'].mean() -
                data.iloc[control_boot,]['revenue_person'].mean())
    bootstrap.append(estimate)

print("The standard deviation is: " + str(np.round(np.std(bootstrap), 2)))

The estimate is: 3.48
The standard deviation is: 0.16
```

2. **(5 points)** Using the second version of the experiment – city-level randomization – estimate the treatment effect and the standard errors. Remember that the units of the analysis in this schematic are not people but cities.
To assess the treatment effect in this experiment, we have to collapse the person-level data into city-level data. Entire cities are the units of the analysis, and they are placed in the treatment or control group entirely; and so we compute the mean revenue for each city and similarly identify whether it is a treated city or control city. Once we have the collapsed data, we implement the same methodology, comparing the average revenue among treated cities to the average revenue among control cities. Note that we cannot use the city as an additional covariate in this exercise, because cities are exactly the unit of analysis.

```
# Collapse the data into city-level units, with treatment and revenue status
city_data = pd.merge(data.groupby('city')['treatment_city'].mean(),
                     data.groupby('city')['revenue_city'].mean(), on = 'city')
```

```
# Now repeat the same analysis as before, except using this dataset
np.random.seed(0)
bootstrap = []
treatment = np.where(city_data['treatment_city'] == 1)[0]
control = np.where(city_data['treatment_city'] == 0)[0]
bootstrap = []

# Compute the difference in means for the sample (as the estimate)
# And then bootstrap, preserving an exact 50-50 split between cities
for i in range(1001):

  if i == 0:
    estimate = (city_data.iloc[treatment,]['revenue_city'].mean() -
                city_data.iloc[control,]['revenue_city'].mean())
    print("The estimate is: " + str(np.round(estimate, 2)))

  else:
    treatment_boot = np.random.choice(treatment, len(treatment), replace = True)
    control_boot = np.random.choice(control, len(control), replace = True)
    estimate = (city_data.iloc[treatment_boot,]['revenue_city'].mean() -
                city_data.iloc[control_boot,]['revenue_city'].mean())
    bootstrap.append(estimate)

print("The standard deviation is: " + str(np.round(np.std(bootstrap), 2)))

The estimate is: 1.15
The standard deviation is: 3.81
```

3. **(5 points)** Compare the two estimates on the basis of bias and variance. Under what conditions would you trust each one more?
   The first estimate has lower variance but we suspect it is more biased, compared to the second estimate, precisely because the first estimate retains the power of person-level randomization at the cost of ignoring network effects. As such, if you suspect network effects are weak, then the first approach is preferable: it has far lower variance but only slightly more bias. But if you suspect network effects are strong, then the second approach is the only one that gives you remotely plausible estimates, even if it has higher variance.

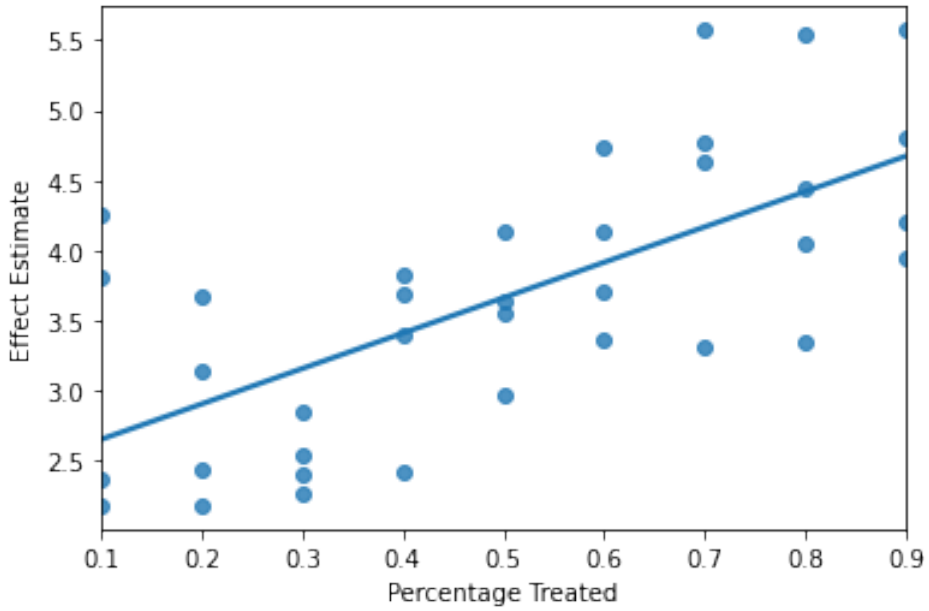
4. In addition to the concerns around bias and variance discussed above, these two estimates only target the total treatment effect; and do not allow us to identify a direct effect from any indirect effects. We will use the two-stage randomization schematic to improve on this. We will make the strong simplifying assumptions that the direct effect is constant, and that the direct and indirect treatment effects are additive.

   (a) **(10 points)** Estimate the treatment effect within each city. Plot the estimated treatment effect against the proportion of the city treated, and fit a straight line to the data.

   ```
   # Collapse the data into treatment intensity, and difference between treatment
   # and control by city
   revenue_treatment = (data.loc[data['treatment_twostage'] == 1]).groupby(
   ```

```
    'city')['revenue_twostage'].mean()
revenue_control = (data.loc[data['treatment_twostage'] == 0]).groupby(
    'city')['revenue_twostage'].mean()
twostage_data = pd.merge(revenue_treatment, revenue_control, on = 'city')
twostage_data['difference'] = (twostage_data['revenue_twostage_x'] -
                 twostage_data['revenue_twostage_y'])
twostage_data = pd.merge(data.groupby('city')['treatment_twostage'].mean(),
                         twostage_data[['difference']], on = 'city')

# Plot the graph and fit a line
import seaborn as sns
import matplotlib.pyplot as plt
plot = sns.regplot(twostage_data['treatment_twostage'],
                   twostage_data['difference'], ci = None)
plot.set(xlabel='Percentage Treated', ylabel='Effect Estimate')
plt.show()
```



(b) **(5 points)** Use the line to estimate the direct treatment effect and the indirect treatment effect on the treated. Since the direct effect is constant, a valid interpretation is that it is the effect of receiving treatment when nobody else in your city does. Furthermore, the indirect effect on the treated is the added effect, for a given treated user, as you move from nobody in your city to everyone in your city receiving treatment.
In fact, the line — whose parameters we can extract by linear regression — precisely gives us these two terms. The intercept tells us the effect of treatment for a user living in a city that has 0% of its population treated, and is thus the direct effect. The slope tells us the incremental effect as more and more of your city is treated, i.e. as the fraction moves from 0 to 1.

```
result = sm.ols(formula = 'difference ~ treatment_twostage',
                data = twostage_data).fit()
```

```
print("The estimate of the direct effect is: " +
      str(np.round(result.params[0], 2)))
print("The estimate of the indirect effect is: " +
      str(np.round(result.params[1], 2)))

The estimate of the direct effect is: 2.4
The estimate of the indirect effect is: 2.53
```

(c) **(5 points)** Bootstrapping for two-stage randomization is fairly complex. Rather than implement it, describe verbally how you would do it.

Bootstrapping must be done in a two-stage process, to mimic the original experiment. First, we must assign cities to treatment percentages randomly, e.g. Atlanta has 30% in the first run, Atlanta has 20% in the second run, etc. There is a subtlety here: for this first stage, we don't re-sample cities (e.g. we don't have two Atlantas), because presumably we have the entire population of cities – rather than a sample – in which we would launch our experiment. (However, because this point is very subtle, I deducted no points for mistaken answers.) Second, for each iteration, we must sample within each city accordingly. In this example, if Atlanta in practice has 60% of its consumers in the treatment group, then in the first bootstrap iteration, we would sample from the 60% treated and 40% control groups to create bootstrapped treatment and control groups of 30% and 70% respectively. We can then estimate the parameters as before.

(d) **(5 points)** Note that standard errors generated when you implement the bootstrap are respectively 0.32 and 0.58. Using this information, compare this estimator to the first two on bias and variance.

This suggests that the two-stage randomization is a large improvement: we get estimates that acknowledge network effects and thus have lower bias than the first estimator (person-level), but while also having much lower variance than the second estimator (city-level). Furthermore, this approach gives us direct and indirect effects, so that we can understand the effects of treatment more richly. The one downside to note is that this is a fairly structural approach, which requires us to make assumptions about the structure of the direct and indirect effects, e.g. constant direct effects and additive effects. If the structure is wrong, this approach will give us misleading results.