Exercise 6: Linear models and machine learning

Kristian Urup Olesen Larsen[1]

[1]kuol@econ.ku.dk

March 23, 2020

Problem 6.1.1 - Wrapping in a function

This first question should be straight forward to complete, we simply wrap the code in a def and return statement

```
def simulate(n = 1000, m = 1500, k = 10, plot = False):
    # [function code]
    return (d, z, Pi, gamma, nu), (y, x, delta, alpha, u)
```

with an if-statement ensuring that the plotting code is only run when plot = True. Notice that the function returns two tuples of data. This is simply to make unpacking results more intuitive, as we can call

```
aux, main = simulate()
d, z, Pi, gamma, nu = aux
y, X, delta, alpha, u = main
```

Now let us quickly discuss what the code is actually doing. The main part of the code is straight forward, it simulates data according to the two equations

$$d_i = x_i'\gamma_0 + z_i'\delta_0 + u_i \tag{Auxilliary}$$
$$y_i = \alpha_0 d_i + x_i'\beta_0 + \epsilon_i \tag{Main}$$

The important part happens in one of the first lines, namely

```
errors = np.random.multivariate_normal(mean = [0,0], cov = [[5, 5], [5, 5]], size = n)
```

which constructs u_i, ν_i as correlated random normals, as evident from figure 1. The consequence of this is that whenever u_i is high we also expect ϵ_i to be high and vice versa. As we will see shortly this becomes an issue if we want to estimate the actual impact of d_i on y_i. i.e. α_0.

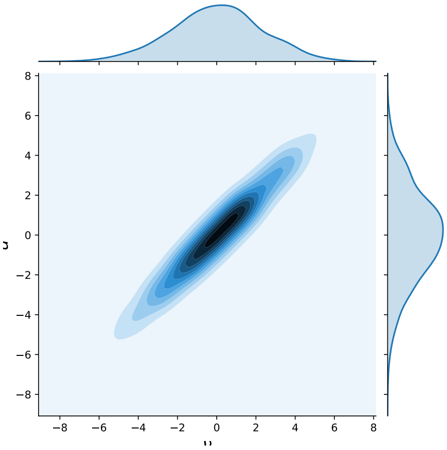

Fig 1. Correlation structure of ν_i and u_i.

Problem 6.1.2 - The naive regression

In this problem we will see what quantity we can expect to estimate by running the regression

$$y_i \sim \beta_0 + \beta_1 d_i + \eta_i \tag{1}$$

If this regression provides unbiased estimates on d_i we will expect $\hat{\beta}_1 \approx \alpha_0$. Repeating the regression many times should reveal that $\hat{\beta}_1$ falls evenly on either side of the true value α_0. However, this is not what we will find, and the reason lies in the correlated errors.

Because u_i and ϵ_i are positively correlated $u_i > 0 \Rightarrow \mathbb{E}[\epsilon_i|u_i] > 0$ - i.e. positive errors in the main equation are associated with positive errors in the auxilliary equation. Assume for a moment that u_i and ϵ_i are correlated such that $\partial\epsilon_i/\partial u_i = \sigma$. Then increasing u_i dicretly increases ϵ_i by a proportional amount. In this case what happens when $u_i = 1$? d_i increases by 1 above $\mathbb{E}[d_i|z_i, x_i]$, and y_i increase by $\alpha_0 + \sigma$ above $\mathbb{E}[y_i|x_i]$. From observing only y_i and d_i we cannot separate out what changes in y_i were due to changes in d_i (α_0), and what changes were due to changes in u_i affecting ϵ_i (σ). For this reason we should expect a bias in our naive regression results. The code for this problem starts off with setting up two lists and a for-loop. The first few lines of the loop simply simulates a new dataset and unpacks all of the return values.

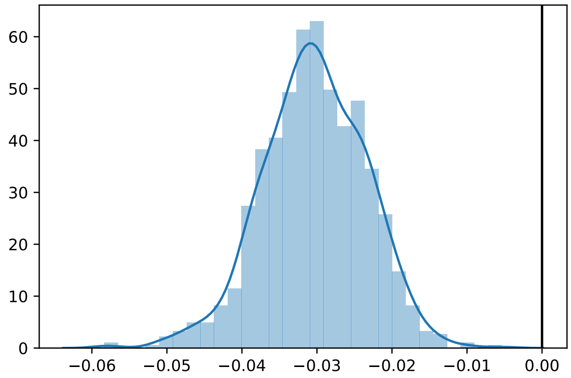

Fig 2. Distribution of estimated values for $\hat{\beta}_1$'s deviation from truth in the naive regression.

```
A,B = [], []
for _ in range(1000):
    aux, main = simulate()
    d, z, Pi, gamma, nu = aux
    y, X, delta, alpha, u = main
```

Next we fit the naive regression

```
    model = OLS(y, add_constant(d))
    result = model.fit()
```

and store the true α_0 alongside $\hat{\alpha}_0$ in the two lists.

```
    A.append(alpha)
    B.append(result.params[1])
```

The histogram can easily be drawn with seaborn,

```
sns.distplot([(a - b) for a,b in zip(A,B)])
plt.axvline(0,color = 'black')
```