Empirical exercise – Implication of multicollinearity for the OLS estimator

1. Set a seed for reproducible results

Set a seed for reproducible results.

```
clear;
rng(1)
```

2. Set the number of simulations

Set the number of simulations to be carried out.

```
N_sim = 1000;
```

3. Set the sample size

Assume a linear regression model with `N_obs` observations available for the variables of this model.

```
N_obs = 1000;
```

4. Set true values for the coefficients of the intercept and the independent variable

Assume that the liner regression model contains a constant term and two independent variables. Assume also that this is the true DGP and that we know the true values of the coefficients of the variables of this model.

```
B_true = [0.2; 0.5; 0.75];
```

5. Create the constant term

Create the constant term.

```
x_0 = ones(N_obs,1);
```

6. Create a vector of covariances between two independent variables

A common efficiency problem is multicollinearity, or correlation between independent variables. Technically, only perfect collinearity between two or more of the independent variables violates an OLS assumption, while any multicollinearity short of that is better thought of as an empirical challenge rather than an assumption violation. Still, when two (or more) independent variables are correlated with each other, the variability in the coefficient estimates operating on those independent variables increases, which leads to increases in the standard errors. As a result, separating the unique effects of each independent variable on the dependent variable becomes more difficult. In this exercise we will study an example of this using simulation. To introduce multicollinearity, we will generate two independent variables that are correlated with each other.

We will examine the implications of multicollinearity for the OLS estimator across a range of correlations between independent variables. Let us consider two independent variables, correlated at 11 different values from 0 to 0.99. The first line of the code at the end of the section creates a vector array containing the different correlation values. We also consider these values as covariances by assuming that each of the two correlated variables has a variance of 1. The second line of the code assigns the size of the column dimension of this vector to a variable named `N_covariance_par`. We will use this variable when iterating over different correlation scenarios in our simulation.

```
covariance_par = [0:0.1:0.9 0.99];
covariance_par_j = size(covariance_par,2);
```

7. Create empty matrices to store the simulated OLS coefficient estimates, standard errors, standard deviation of the simulated OLS coefficient estimates

The MATLAB code at the end of the section creates a number of empty matrices. In particular, in the second line, `B_hat_x_1_sim` is to store coefficient estimates of the independent variable `x_1` from `N_sim` simulations.

In the third line, `B_hat_x_1_sim_j` is to store coefficient estimates of the independent variable `x_1` from `N_sim` simulations at `N_covariance_par` different correlation levels between the independent variables `x_1` and `x_2`. That is `B_hat_x_1_sim_j` collects in its columns `B_hat_x_1_sim` from different correlation scenarios. The suffix `j` in the name of the matrix arrray `B_hat_x_1_sim_j` is referring to the index of the for loop we will consider that will iterate over 11 different correlation scenarios.

In line four, `B_hat_x_1_SE_sim` is to store the standard error of the coefficient estimate of `x_1` from a given simulation of the coefficient estimate of the independent variable `x_1`. Since we conduct `N_sim` simulations, `B_hat_x_1_SE_sim` will contain `N_sim` standard errors.

In line five, `B_hat_x_1_SE_sim_j` collects in its columns `B_hat_x_1_SE_sim` from different correlation scenarios.

In line six, `B_hat_x_1_sim_j_SD` is to store the standard deviation of the `N_sim` simulated coefficient estimates of `x_1` from each of the `N_covariance_par` different correlation scenarios. Note that we are not saving a quantity for each individual coefficient estimate of `x_1` from a given simulation. We are saving one summary statistic from `N_sim` simulations at each correlation value.

```
N_par = 1;
B_hat_x_1_sim = NaN(N_sim,N_par);
B_hat_x_1_sim_j = NaN(N_sim,covariance_par_j);
B_hat_x_1_SE_sim = NaN(N_sim,N_par);
```

```
B_hat_x_1_SE_sim_j = NaN(N_sim,covariance_par_j);
B_hat_x_1_sim_j_SD = NaN(covariance_par_j,N_par);
```

8. Define input arguments for the multivariate normal random number generator

In the next section, we will draw random numbers from the multivariate normal distribution to create two independent variables that are correlated with each other. To do this, we will make use of the built-in MATLAB function `mvnrnd`. The function accepts three input arguments. The first input argument is the mean vector of the distribution. The second input argument is the covariance matrix of the distribution. The third input argument specifies the number of observations to be drawn for each random variable of the distribution. Here we define the first and the third input arguments. The second input argument is to be defined within the simulation in the next section because in each iteration of the simulation the covariance matrix will be updated in accordance with the different levels of correlation between two independent variables.

```
MU = [0 0];
cases = N_obs;
```

9. Create sampling distributions for the OLS coefficient estimates under different correlation levels between the independent variables

At the end of the section we consider two for loops. The inner for loop simulates `N_sim` coefficient estimates from repeated sampling. The outer for loop repeats this simulation for 11 different correlation levels between two independent variables.

Consider the inner for loop. In the first line of the for loop we define the index of the for loop. In the second line, we generate random values for the error term. Note that we rule out heteroskedasticity by setting the standard deviation of the error term to 1. In the third line, we construct the systematic component of the regression equation. In line four, we generate values for the dependent variable. In line five, we call the function `exercisefunction` to estimate the coefficient of `x_1` and the standard error of this coefficient estimate in each of the `N_sims` random samples we draw in the inner for loop. In lines six and seven, we collect the simulated coefficient estimates in the vector array `B_hat_x_1_sim(i,1)`, and the simulated standard errors of the coefficient estimates in the vector array `B_hat_x_1_SE_sim(i,1)`.

Consider the outer for loop. In the first line of the for loop we define the index of the for loop. In the second line we define the covariance matrix of the two independent variables of the regression model. The covariance matrix is updated in each iteration of the for loop using the covariance values taken from the range of values contained in the vector array `covariance_par`. We set the variances of each independent variable to 1. In the third line, we supply the `mvnrnd` function with the defined covariance matrix, and with two other input arguments defined in the previous section. The function generates random values for the two independent variables from the multivariate normal distribution so that the variables are correlated. In the fourth and fifth lines of the for loop we define the independent variables of the model (`x_1` and `x_2`).

```
for j = 1:covariance_par_j
```

```
    SIGMA = reshape([1 covariance_par(:,j) covariance_par(:,j) 1],2,2);
    x_1_x_2_correlated = mvnrnd(MU,SIGMA,cases);
    x_1 = x_1_x_2_correlated(:,1);
    x_2 = x_1_x_2_correlated(:,2);
    for i = 1:N_sim
        u = normrnd(0,1,N_obs,1);
        X = [x_0 x_1 x_2];
        y = X*B_true+u;
        LSS = exercisefunction(y,X);
        B_hat_x_1_sim(i,1) = LSS.B_hat(2,1);
        B_hat_x_1_SE_sim(i,1) = LSS.B_hat_SE(2,2);
    end
    B_hat_x_1_sim_j(:,j) = B_hat_x_1_sim(:,1);
    B_hat_x_1_sim_j_SD(j,:)  = std(B_hat_x_1_sim_j(:,j));
    B_hat_x_1_SE_sim_j(:,j) = B_hat_x_1_SE_sim(:,1);
end
```

10. Plot the sampling distributions of the OLS coefficient estimates at different correlation levels between the independent variables

The plot shows the sampling distribution of the OLS coefficient estimate of `x_1` when the correlation between `x_1` and `x_2` is 0, and when it is 0.99. The comparison of the two distributions demonstrates how a high correlation between the two independent variables affects the OLS estimator. The OLS estimator performs worse in estimating the true value when there is correlation between the independent variables.

```
ksdensity(B_hat_x_1_sim_j(:,1))
hold on
ksdensity(B_hat_x_1_sim_j(:,11))
hold on
line([mean(B_hat_x_1_sim_j(:,1)) mean(B_hat_x_1_sim_j(:,1))],ylim,'Color','black')
hold on
line([mean(B_hat_x_1_sim_j(:,11)) mean(B_hat_x_1_sim_j(:,11))],ylim,'Color','black')
hold off
title('The Effect of Multicollinearity on the Distribution of the OLS Estimator')
legend('No correlation','Almost perfect correlation','B_hat_x_1_sim_mean')
ylabel('Density')
xlabel('B_hat_x_1')
```

11. Plot the standard deviation of the sampling distribution of the OLS coefficient estimates against different correlation levels between the independent variables

The plot shows the standard deviation of 1000 simulated coefficient estimates at each of 11 different correlation scenarios. The standard deviation increases as the correlation between the independent variables increases. However, notice that the rate of increase is not linear. It is a small increase from correlations of 0 to 0.7, then larger increases from 0.7 to 0.99. This tells us that efficiency loss as a result of multicollinearity is relatively minor for most of the range of correlation levels, but becomes a more serious problem when independent variables are highly correlated.

Increase the sample size `N_obs` from 1000 to, say, 5000. Inspect the change in the plot. What do you conclude? We obtain a similar pattern as with 1000 observations. However, the problem is less severe: the standard deviation of the estimates at a correlation of 0.99 is less than half of the value it was when `N_obs` was 1000. This demonstrates one major solution for multicollinearity: collect more data. The multicollinearity is a problem because it reduces the amount of information available to estimate your coefficients. Increasing the sample size compensates for this by adding more information in the form of more data points back into the analysis.

```
plot(covariance_par,B_hat_x_1_sim_j_SD)
title('Standard Deviation of the Simulated OLS Estimates at Different Correlation
Levels Between the Independent Variables')
ylabel('S.D. of B_hat_x_1')
xlabel('Correlation Between the Independent Variables')
```

12. Plot the sampling distributions of the standard error estimates of the OLS coefficient estimates at different correlation levels between the independent variables

Consider one of the two sampling distributions of the OLS estimator we have produced in Section 10. This distribution gives some idea about the standard error of the OLS estimator. A formal estimator of the standard error of the OLS estimator is given by `LSS.B_hat_SE` in `exercisefunction.m`. In Section 10 (or in 11) we have seen that multicollinearity increases the sampling variance of the the OLS estimator. The formal estimator of the standard error of the OLS estimator should reflect this. Produce the plot using the code at the end of the section. The plot shows that the standard error of the OLS coefficient estimate of `x_1` indeed increases when `x_1` is correlated with another independent variable. Observe also that the sampling variance of the standard error estimator also increases as we increase the correlation level between the independent variables.

```
ksdensity(B_hat_x_1_SE_sim_j(:,1))
hold on
ksdensity(B_hat_x_1_SE_sim_j(:,11))
title('The Effect of Multicollinearity on the Standard Error of the OLS Estimator')
legend('No correlation','Almost perfect correlation')
ylabel('Density')
xlabel('B_hat_x_1_SE')
```