Exercise – Understanding multicollinearity using simulation

1. Aim of the exercise

Econometrics classes discuss the implications of multicollinearity theoretically. This exercise uses simulation to illustrate the implications of multicollinearity for the sampling distribution of the OLS estimator for a clearer understanding of the implications of multicollinearity.

2. Theory

Multicollinearity is a common efficiency problem that occurs when two or more explanatory variables in a regression model are highly correlated. When explanatory variables are strongly correlated, they share a significant amount of common information, which makes it challenging to estimate their individual effects accurately. This high degree of correlation among explanatory variables leads to inflated variances for the estimated regression coefficients, resulting in unstable and less reliable estimates.

3. Application

3.1. Clear the memory

Clear the memory from possible calculations from an earlier session.

```
% 3.1. Clear the memory
clear;
```

3.2 Set the number of simulations

Set the number of simulations to be carried out.

```
% 3.2. Set the number of simulations
N_sim = 1000;
```

3.3. Set the sample size

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

```
% 3.3. Set the sample size
N_obs = 1000;
```

3.4. Set true values for the coefficients of the model

Assume that the liner regression model contains two independent variables and, for simplicity, no constant. Assume also that this is the true DGP and that we know the true values of the coefficients.

```
% 3.4. Set true values for the coefficients
B_true = [0.5 0.75]';
```

3.5. Create the constant term

Create the constant term.

```
23 % 3.5. Create the constant term
24 x_0 = ones(N_obs,1);
```

3.6. Create a vector of covariances between two independent variables

Our aim is to examine the implications of multicollinearity for the OLS estimator across a range of correlations between independent variables. Consider two correlated independent variables. We define correlation levels using 10 different covariance values, ranging from 0 to 0.99. Line 2 creates a vector array containing these values. We assume that each variable has a variance of 1. The next line defines the column dimension of this vector. We will use this variable when iterating over different correlation scenarios in our simulation.

```
26 % 3.6. Create a vector of covariances between two independent variables
27 sigma_x_1_x_2 = 0:0.11:0.99;
28 N_sig = size(sigma_x_1_x_2,2);
```

3.7. Define the number of coefficients to be simulated

Define the number of coefficients to be simulated.

```
30 % 3.7. Define the number of parameters to be estimated
31 N_par = 1;
```

3.8. Preallocate matrices for storing simulated statsitics

Here we preallocate several matrices. `B_hat_1_sim` will store coefficient estimates of the independent variable `x_1` from `N_sim` simulations. `B_hat_1_sim_j` will store coefficient estimates of the independent variable `x_1` from `N_sim` simulations at `N_sig` different correlation levels between the independent variables `x_1` and `x_2`. That is, it collects in its columns `B_hat_1_sim` from different correlation scenarios. The suffix `j` refers to the index of the for loop that we will consider to iterate over 10 different correlation scenarios. `B_hat_1_SE_sim` will store the standard error estimates of the coefficient estimates of `x_1` from simulations of the coefficient. Since we conduct `N_sim` simulations, `B_hat_1_SE_sim` will contain `N_sim` standard error estimates. `B_hat_1_SE_sim_j` will collect in its columns `B_hat_1_SE_sim` from different correlation scenarios. `B_hat_1_sim_j_SD` will store the standard deviation of the `N_sim` simulated coefficient estimates of `x_1` from each of the `N_sig` 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 level.

```
33 % 3.8. Preallocate matrices for storing statistics to be simulated
34 B_hat_1_sim = NaN(N_sim,N_par);
35 B_hat_1_sim_j = NaN(N_sim,N_sig);
36 B_hat_1_SE_sim = NaN(N_sim,N_par);
37 B_hat_1_SE_sim_j = NaN(N_sim,N_sig);
38 B_hat_1_sim_j_SD = NaN(N_sig,N_par);
```

3.9. Define an input argument 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 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 input argument. 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. The third inout argument is already defined.

```
40 % 3.9. Define mean vector for the multivariate random number generator
41 mu = [0 0];
```

3.10. Create sampling distributions for the OLS coefficient estimates at different correlation levels between the independent variables

Here we consider a nested loop structure, with one outer loop and one inner loop. The inner for loop simulates `N_sim` coefficient estimates from repeated sampling. The outer for loop repeats this simulation for 10 different correlation levels between two independent variables.

Lines 2 and 3 define the indexes of the for loops. Line 4 defines 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 `N_sig`. We set the variances of each independent variable to 1. Line 6 supplies the `mvnrnd` function with the defined covariance matrix, and with two other input arguments already defined. The function generates random values for the two independent variables from the multivariate normal distribution so that the variables are correlated. Lines 7 and 8 define the independent variables of the model (`x_1` and `x_2`). Line 9 constructs the systematic component of the regression equation. Line 10 generates random values for the error term. Note that we rule out heteroskedasticity by setting the standard deviation of the error term to 1. Line 11 generates values for the dependent variable. Line 12 calls 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. Lines 13 and 15 collect the simulated coefficient estimates in the vector array `B_hat_1_sim(i,1)`, and the simulated standard errors of the coefficient estimates in the vector array `B_hat_1_SE_sim(i,1)`.

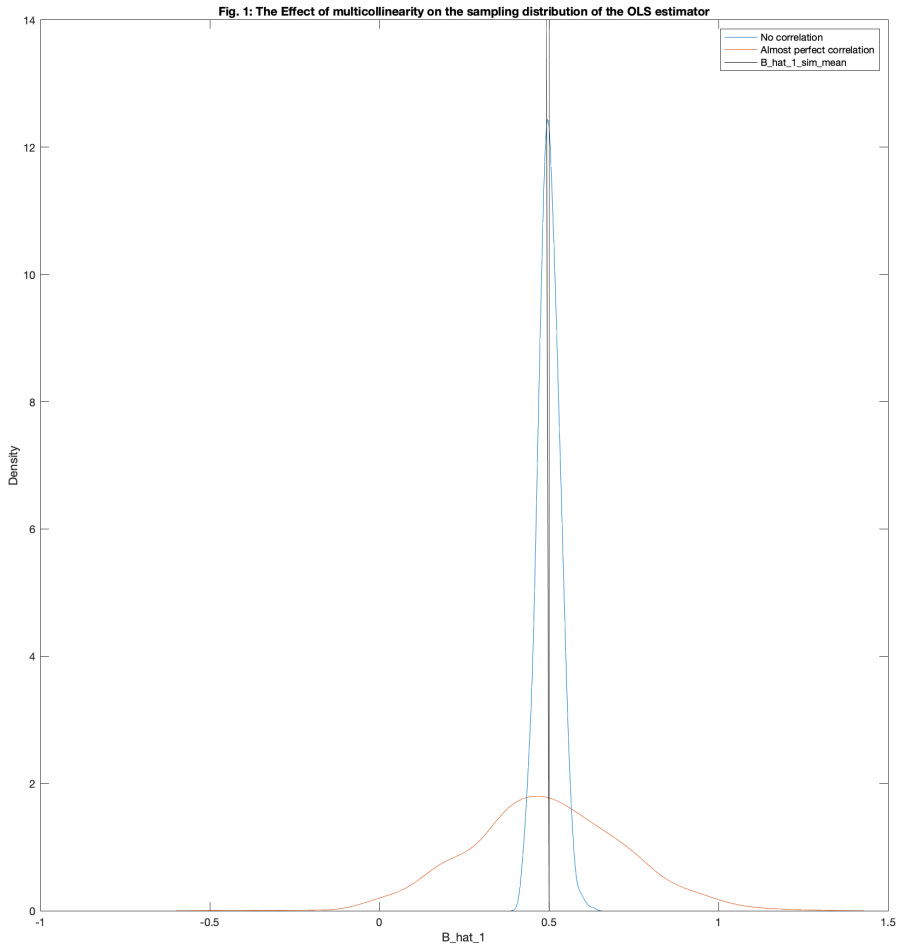
```
43 % 3.10. Create the sampling distribution of the OLS esimator
44 for j = 1:N_sig
45     for i = 1:N_sim
46         Sigma = reshape([1 sigma_x_1_x_2(:,j) sigma_x_1_x_2(:,j) 1], ...
47             2,2);
48         x_1_x_2_mvn = mvnrnd(mu,Sigma,N_obs);
49         x_1 = x_1_x_2_mvn(:,1);
50         x_2 = x_1_x_2_mvn(:,2);
51         X = [x_1 x_2];
52         u = random('Normal',0,1,[N_obs 1]);
53         y = X*B_true+u;
```

```
        LSS = exercisefunctionlss(y,X);
        B_hat_1_sim(i,1) = LSS.B_hat(1,1);
        B_hat_1_sim_j(:,j) = B_hat_1_sim(:,1);
        B_hat_1_SE_sim(i,1) = LSS.B_hat_SEE(2,1);
        B_hat_1_SE_sim_j(:,j) = B_hat_1_SE_sim(:,1);
        B_hat_1_sim_j_SD(j,:) = std(B_hat_1_sim_j(:,j));
    end
end
```

4. 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.

```
%% 4. Plot the sampling distribution of the OLS estimator
ksdensity(B_hat_1_sim_j(:,1))
hold on
ksdensity(B_hat_1_sim_j(:,10))
hold on
line([mean(B_hat_1_sim_j(:,1)) mean(B_hat_1_sim_j(:,1))],ylim, ...
    'Color','black')
hold on
line([mean(B_hat_1_sim_j(:,9)) mean(B_hat_1_sim_j(:,10))],ylim, ...
    'Color','black')
hold off
title(['Figure 1: The Effect of multicollinearity on the sampling ' ...
    'distribution of the OLS estimator'])
legend('No correlation','Almost perfect correlation', ...
    'B\_hat\_1\_sim\_mean')
ylabel('Density')
xlabel('B\_hat\_1')
```



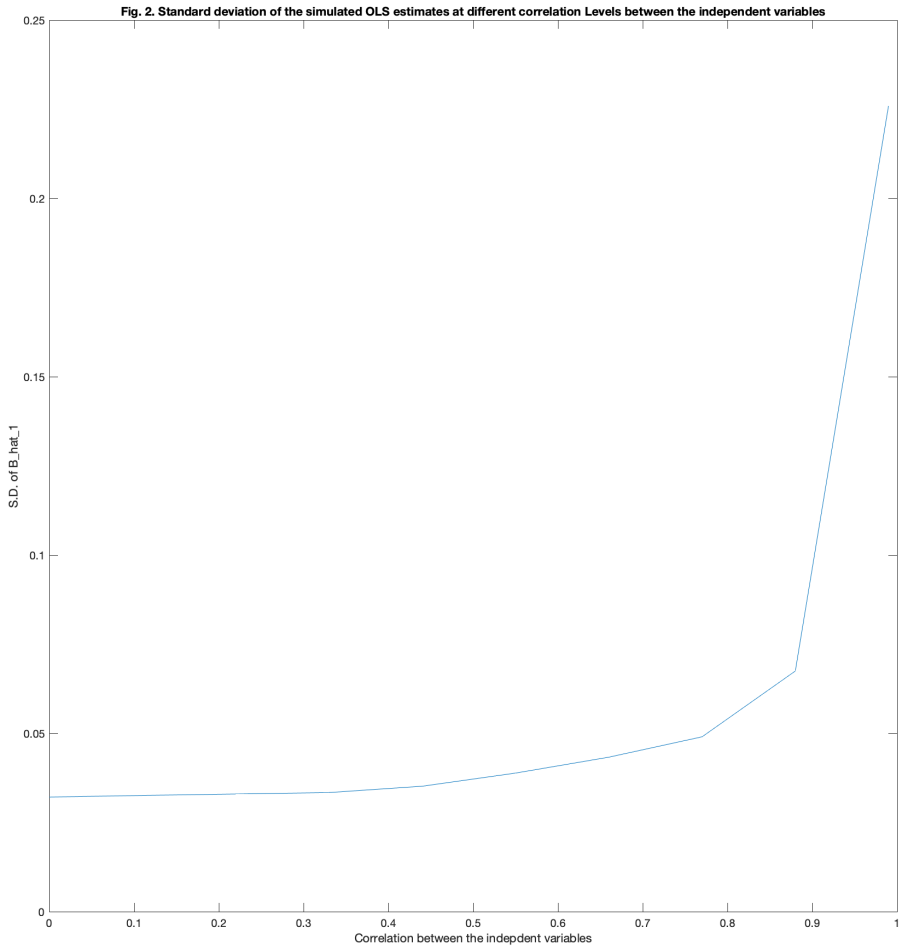
5. Plot the standard deviation of the sampling distribution of the OLS coefficient estimates against different correlation levels between the independent variables

The plot shows the standard deviation of 1000 simulated coefficient estimates at each of 10 different correlation scenarios. The standard deviation increases as the correlation between the independent variables increases. The rate of increase is not linear. 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. 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 the coefficients. Increasing the sample size compensates for this by adding more information in the form of more data points back into the analysis.

```
81 %% 5. Plot the SD of the sampling distribution of the OLS estimator
82 plot(sigma_x_1_x_2,B_hat_1_sim_j_SD)
```

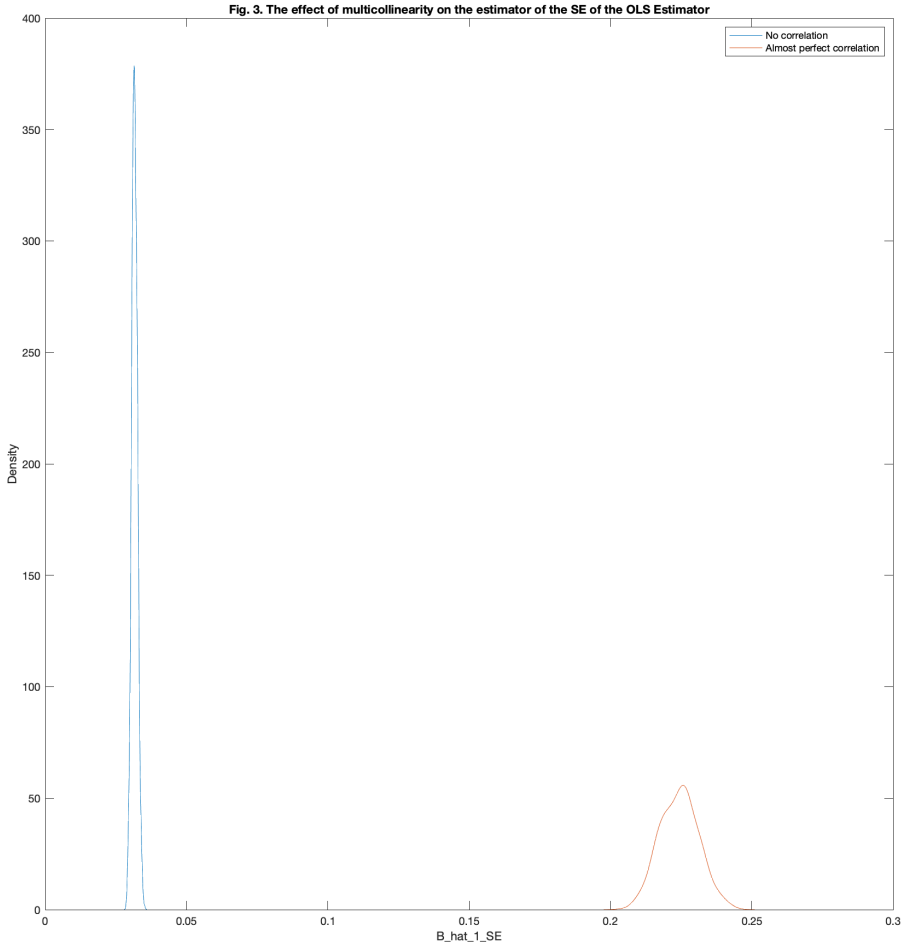
```
title(['Fig. 2. Standard deviation of the sampling distribution ' ...
    'of the OLS estimator at different correlation Levels between ' ...
    'the independent variables'])
ylabel('S.D. of B\_hat\_1')
xlabel('Correlation between the indepdent variables')
```



6. 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 produced above. This distribution gives an 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 the accompanying file `exercisefunction.m`. We saw that multicollinearity increases the variance of the OLS estimator. The formal estimator of the standard error of the OLS estimator should reflect this. The plot here 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 variance of the standard error estimator also increases when the correlation level between the independent variables increases.

```
%% 6. Plot the sampling distribution of the SE estimator
ksdensity(B_hat_1_SE_sim_j(:,1))
hold on
ksdensity(B_hat_1_SE_sim_j(:,10))
title(['The effect of multicollinearity on the estimator of the ' ...
    'SE of the OLS Estimator'])
legend('No correlation','Almost perfect correlation')
ylabel('Density')
xlabel('B\_hat\_1\_SE')
```



7. Final notes

This file is prepared and copyrighted by Tunga Kantarcı. Parts of the simulation exercise are based on Carsey, T. M., and Harden, J. J., 2014. Monte Carlo simulation and resampling methods for social science. SAGE Publications. This file and the accompanying MATLAB files are available on GitHub and can be accessed via this link.