GOVT6029:
Advanced Regression Analysis

Problem Set 3

Due date: April 30

Sergio I Garcia-Rios

Investigating the Properties of Linear Regression Using Simulation

The purpose of this homework is to provide a guided, hands-on tour through the properties of the least squares estimator, especially under common violations of the Gauss-Markov assumptions. We will work through a series of programs which use simulated data – *i.e.*, data created with known properties – to investigate how these violations affect the accuracy and precision of least squares estimates of slope parameters. Using repeated study of simulated datasets to explore the properties of statistical models is called Monte Carlo experimentation.[1]

Although you will not have to write much `R` code, you will need to read through the provided programs carefully to understand what is happening. (For this assignment only, I ask that you *not* attach your code, since you will be making only small changes in long programs. You may find it helpful to show just those few lines of code which you changed. Likewise, report only those results and figures needed to answer the questions asked below.)

Getting Acquianted with the Basic Simulation Code

Open the file `mcls.r`, and read through the code carefully. I recommend thinking through what is happening line by line, perhaps even running pieces of the code and checking for yourself what variables have been created (e.g., by `print`ing the variables in memory).

You will note several new commands, such as `for`, `rnorm()`, `mvrnorm()`, etc. A brief guide to these functions appears in Table 1.

[1] Monte Carlo experiments always produce the same results as analytic proofs for the specific case considered. Each method has advantages and disadvantages: proofs are more general and elegant, but are not always possible. MC experiments are much easier to construct and can always be carried out, but findings from these experiments only apply to the specific scenario under study. Where proofs are available, they are generally preferable to MC experiments, but proofs of the properties of more complicated models are sometimes impossible or impractically difficult. This is almost always the case for the properties of models applied to small samples of data. Here, we use Monte Carlo not out of necessity but for pedagogical purposes, as a tool to gain a more intuitive and hands-on understanding of least squares and its properties.

Table 1: *New `R` functions used in this homework.*

Command	Effect
`for (i in a:b) {}`	Loop over the commands in `{}` once for each element in the sequence `a:b`. On each iteration of the loop, increment `i` by one.
`rnorm(n)`	Take `n` draws from the standard Normal distribution, which has mean zero and standard deviation one. To get draws from a Normal with mean `mu` and standard deviation `sigma`, use `mu + sigma*rnorm(n)`
`mvrnorm(n,mu,Sigma)`	Function from the `MASS` library. Take `n` draws from the Multivariate Normal with means given by the vector `mu` and variance-covariance matrix given by `Sigma`
`apply(x,2,mean)`	Calculate the means of each column of `x` and return them as a vector. (This function works generally; to "apply" a different function, just change `mean` to the desired function; to apply that function over rows instead of columns, change `2` to `1`.)
`density(x)`	Calculate a smoothed histogram of `x`, which can then be `plot()`ed.
`expression(math)`	Used to plot mathematical notation; see help for the command `text()` for examples of syntax of `math`

Careful study will reveal that `mcls.r` works through four steps:

1. Set up the joint distribution of $\mathbf{x}_1$, $\mathbf{x}_2$, and $\mathbf{x}_3$, which are multivariate normal with means $\boldsymbol{\mu}_X$ (denoted `muX`) and variance-covariance matrix $\boldsymbol{\Sigma}_X$ (denoted `SigmaX`).[2] Defines the true values of $\boldsymbol{\beta}$ (denoted `b`) and σ (denoted `sigma`), to be used in step 2.

2. Loop over `sims` simulation runs, at each iteration drawing `n` observations of $\mathbf{x}_1$, $\mathbf{x}_2$, and $\mathbf{x}_3$ (collected in the matrix `X`), from which $\mathbf{y}$ (denoted `y`) is generated as:

$$y_i = \beta_0 + \beta_1 x_{1i} + \beta_2 x_{2i} + \beta_3 x_{3i} + \epsilon_i$$

   where ϵ_i is a draw from the Normal distribution with mean 0 and variance σ^2.

3. At the end of each simulation run, find the least squares estimates of the β's above, and save them, along with their standard errors and t-statistics.

4. After the simulation runs are complete, print the average estimates, standard errors, and t-statistics, comparing each to the "true" values. Then plot the distributions of the estimated $\hat{\beta}$'s, again comparing to the truth.

An example will help explain the output from step 4. If we run `mcls.r` at its default settings (which correspond to an ideal case in which the Gauss-Markov assumptions hold, and there is no omitted variable or selection bias), we obtain the following text output:

```
True parameters
    1          2          3          4

Average LS estimate across 1000 simulation runs
(Intercept)          X1          X2          X3
   1.000479    1.992530    3.000542    3.989724
```

The above shows that on average across 1000 simulations, linear regression recovered the true values of the intercept and three slope coefficients almost exactly, despite not knowing these true values. Linear regression works, and without bias, at least under ideal conditions.

Of course, in any particular regression, our estimates may be off from their expected values. The standard error is an estimate of how far off we can expect regression estimates to be – but is the standard error itself well estimated? It should match the standard deviation of estimates of β across different samples of data from the same distribution.

```
True standard errors across 1000 simulation runs
(Intercept)          X1          X2          X3
  0.1408317   0.1447928   0.1417090   0.1491686

Average estimated standard errors across 1000 simulation runs
```

[2] That is, we create a set of covariates which are jointly Normal, $\mathbf{X} = \mathcal{MVN}(\boldsymbol{\mu}_X, \boldsymbol{\Sigma}_X)$.

```
(Intercept)          X1          X2          X3
  0.1435155   0.1451408   0.1448281   0.1452036
```

Comparing the average standard error with the "true" standard deviation across $\hat{\beta}$'s shows that under ideal conditions, linear regression also produces unbiased estimates of its own error.

Because the t-statistic is just the ratio of the estimated β to its standard error, it too should be well estimated on average – as we can see below.

```
[1] "True t-stat across 1000 simulation runs"
(Intercept)          X1          X2          X3
   7.100677   13.812843   21.170141   26.815302
[1] "Average estimated t-stat across 1000 simulation runs"
(Intercept)          X1          X2          X3
   6.971224   13.728258   20.717954   27.476750
```

We also receive Figure 1 as a `pdf` file.

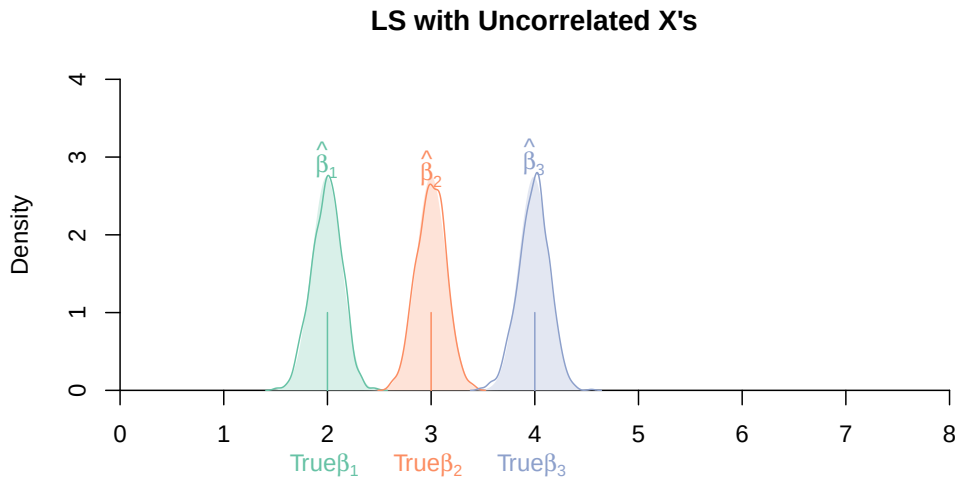

Figure 1: *The true and estimated linear regression coefficients under ideal conditions across 1000 simulated datasets.* Vertical marks indicate the true coefficients used to generate the 1000 datasets. The distribution of least squares estimated coefficients across the 1000 datasets are shown as shaded regions. The distribution of estimates implied by the average estimated standard error is superimposed as a solid line, and matches the actual distribution of estimates almost exactly.

Moving Beyond the Default Simulation Settings

The default settings in `mcls.r` create three uncorrelated covariates (note the default `SigmaX`) and generate **y** from them using the "true" model,

$$y_i = 1 + 2x_{1i} + 3x_{2i} + 4x_{3i} + \epsilon_i, \qquad \epsilon_i \sim \mathcal{N}(0, 2).$$

Then we attempt to recover these specific true β's by regressing $\mathbf{y}$ on $\mathbf{x}_1$, $\mathbf{x}_2$, and $\mathbf{x}_3$.

We will use `mcls.r` as a template to explore when regression works and when and how it fails. By changing the settings in the first and second part of the code, we can estimate the linear regression model using different types of data, and see the consequences of different data problems on estimation bias and efficiency. By changing the third part of the code, we can change the model used, to compare the performance of different least squares models applied to the same data. You will be provided alternative versions of the code to accomplish this, but will also be asked to make some changes to the code on your own.

Problems to Solve

Now that we have read through the code, we are ready to begin.

a. Run `mcls.r` using its default settings. Make a note of the results. Rerun the program three times, setting the correlation of $\mathbf{x}_1$ and $\mathbf{x}_2$ to 0.5, 0.9, and 0.99, respectively.[3] Based on the results from these runs, what can you say about the effect of partial collinearity on least squares estimates? In particular, does raising the correlation of $\mathbf{x}_1$ and $\mathbf{x}_2$ add bias to our estimates of β_1, β_2, or β_3? Does raising the correlation of $\mathbf{x}_1$ and $\mathbf{x}_2$ affect the precision of estimates of β_1, β_2, or β_3?

b. Set the correlation of $\mathbf{x}_1$ and $\mathbf{x}_2$ to 1, and rerun `mcls.r`. What has happened, and why? It will help to look at the summary of the regression results for the last run, using `print(summary(res))`.

c. Now open the program `mcovb.r` in your text editor. Note that this program is identical to `mcls.r`, with one exception. When this program runs `lm()`, it omits $\mathbf{x}_2$ from the regression. Now run the program at its default settings, with the correlation of $\mathbf{x}_1$ and $\mathbf{x}_2$ set to 0. What effect does the omission of $\mathbf{x}_2$ have on the bias and precision of the estimates of β_1 and β_3?

d. Set the correlation of $\mathbf{x}_1$ and $\mathbf{x}_2$ to 0.9, and rerun `mcovb.r`. Now what effect does the omission of $\mathbf{x}_2$ have on the bias and precision of the estimates of β_1 and β_3? Do our findings differ from those in part **c**? Why?

e. Finally, keep the correlation of $\mathbf{x}_1$ and $\mathbf{x}_2$ at 0.9, but rewrite `mcovb.r` to run the regression of $\mathbf{y}$ on $\mathbf{x}_1$ and $\mathbf{x}_2$, omitting $\mathbf{x}_3$. What effect does the omission of $\mathbf{x}_2$ have on the bias and precision of the estimates of β_1 and β_2?

f. What explains the differences in your results across parts **c**, **d**, and **e**? Based on these results, and your findings in part **a**, how would you recommend users of least squares deal with highly correlated covariates?

g. Open the program `mcselect.r` in your text editor. Note that this program is identical to `mcls.r`, except now, all observations in which $\mathbf{y}$ is greater than its sample mean are deleted prior to running the regression. What effect does selection on $\mathbf{y}$ have on the bias and precision of the estimates of β_1, β_2, and β_3?

h. Open the program `mchet.r` in your text editor. Note that this program is identical to `mcls.r`, except the structure of `sigma` has changed. In this simulation, we will assume the data y_i are Normally distributed such that

$$\begin{aligned} y_i &\sim \mathcal{N}(\mu_i, \sigma_i^2), \\ \mu_i &= \beta_0 + \beta_1 x_{1i} + \beta_2 x_{2i} + \beta_3 x_{3i}, \text{and} \\ \sigma_i^2 &= \exp(\gamma_0 + \gamma_1 x_{1i}). \end{aligned}$$

[3] Be careful that you set `SigmaX` to possible values only. This matrix must always be symmetric, so to set the covariance of $\mathbf{x}_1$ and $\mathbf{x}_2$ to 0.5, you must set both `SigmaX[2,1]` and `SigmaX[1,2]` to 0.5.

That is, our data are heteroskedastic. (The γ's are set in a vector called `g`.)

Run `mchet.r` under its default setting, which sets $\gamma_0 = \log(2)$ and $\gamma_1 = 0$. Confirm that under these settings, $\mathbf{y}$ is still homoskedastic. Note the result. Now try adding heteroskedasticity by increasing γ_1 to 1. Confirm that changing this setting has made $\mathbf{y}$ heteroskedastic. What effect does this added heteroskedasticity have on our results?

i. Open the program `mcautocor.r` in your text editor. Note that this program is identical to `mcls.r`, except for two differences. First, $\mathbf{y}$ now depends on the present and past error term:

$$y_i = \beta_0 + \beta_1 x_{1i} + \beta_2 x_{2i} + \beta_3 x_{3i} + \rho\epsilon_{i-1} + \epsilon_i.$$

This is a *moving average* process of order 1, or MA(1). If $\rho \neq 0$, the y_i's will be serially correlated.

Second, the present value of the kth covariate, $x_{k,i}$, now depends on the random part of the past value of the covariate, $x_{k,i-1}$, such that

$$x_{k,i} = \mu_{\mathbf{X}_k} + \rho_{\mathbf{X}_k}\epsilon_{\mathbf{X}_{k,i-1}} + \epsilon_{\mathbf{X}_{k,i}}.$$

This is also a *moving average* process of order 1, or MA(1). If $\rho_{\mathbf{X}_k} \neq 0$ for some k, that $\mathbf{x}_k$ will also be serially correlated.

Run `mcautocor.r` under its default settings, with $\rho = 0$ and $\rho_{\mathbf{X}_k} = 0$ for all covariates k. Note the results. Rerun it twice: first set $\rho = 0.5$ and $\rho_{\mathbf{X}_k} = 0.5$ for all k; then set $\rho = 0.9$ and $\rho_{\mathbf{X}_k} = 0.9$ for all k. Based on the results from these runs, what can you say about the effect of serial correlation on least squares estimates? Experimenting further, what happens if you have serial correlation in $\mathbf{y}$ but not in $\mathbf{X}$, or *vice versa*?

j. Come up with a question about the properties of least squares to investigate using one or more of the provided programs, or modifications thereof. Illustrate the answer to your question by running the program(s) under different settings, and comparing results.

An example question:

> *Which of the problems identified in this homework can be mitigated by gathering more data (e.g., by setting* `n=1000`*, instead of* `n=100`*), and which problems will stay just as severe no matter how much data are collected?*

You are welcome to answer the example question for full credit, but will receive bonus points for formulating your own.