# Communication with External Programs Using R

Sunthud Pornprasertmanit

June 16, 2015

In the previous lecture, we talked about how to use the if statement and the `apply` function. In this class, we will cover how to use R to generate scripts for other programs, run the scripts using other programs, and combine the outputs.

The program for data analysis used in this class is *Mplus* (for TTU people, it is available in TechStat). *Mplus* is designed for statistics in social sciences including structural equation models and multilevel models. Let's use *Mplus* to run a multiple regression model for the `attitude` data. Before doing that, let's run the multiple regression in R first:

```
> out <- lm(rating ~ complaints + privileges + learning + raises +
+     critical + advance, data = attitude)
> summary(out)

Call:
lm(formula = rating ~ complaints + privileges + learning + raises +
    critical + advance, data = attitude)

Residuals:
     Min       1Q   Median       3Q      Max
-10.9418  -4.3555   0.3158   5.5425  11.5990

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) 10.78708   11.58926   0.931 0.361634
complaints   0.61319    0.16098   3.809 0.000903 ***
privileges  -0.07305    0.13572  -0.538 0.595594
learning     0.32033    0.16852   1.901 0.069925 .
raises       0.08173    0.22148   0.369 0.715480
critical     0.03838    0.14700   0.261 0.796334
advance     -0.21706    0.17821  -1.218 0.235577
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 7.068 on 23 degrees of freedom
Multiple R-squared:  0.7326,        Adjusted R-squared:  0.6628
F-statistic:  10.5 on 6 and 23 DF,  p-value: 1.24e-05
```

Six variables from Columns 2–7 of the `attitude` data are used to predict the first variable, `rating`. To use *Mplus*, we need to write the data set to a file first. Then, *Mplus* can take data with comma-separated values (.csv) or tab-delimited file (.txt). The file must not contain column names or row names. The variable names will be specified later in a *Mplus* script. We will use the `write.table` function to save data to a file:

```
> write.table(attitude, file = "attitude.csv", row.names = FALSE,
+     col.names = FALSE, sep = ",")
```

We need to check the directory that this file is saved. The current directory that R is running can be shown by using `getwd()`. All files in the current directory can be listed by `list.files()`. You will see `"attitude.csv"` in the current directory.

```
> getwd()

[1] "C:/Users/spornpra/Desktop"

> list.files()

 [1] "attitude.csv"                    "class4.Rnw"
 [3] "class4.tex"                      "desktop.ini"
 [5] "IBM SPSS Statistics 20.lnk"      "Mplus Editor.lnk"
 [7] "R x64 3.1.0.lnk"                 "RStudio.lnk"
 [9] "SAS 9.4 (English).lnk"           "StatTransfer Twelve (64 Bit).lnk"
[11] "Syntax N 1000"
```

Next, let's write the *Mplus* script to run the multiple regression model. The `write` function will save a text object in R into a file:

```
> myscript <- "
+   TITLE: Multiple Regression on Attitude Data;
+   DATA:
+     FILE IS attitude.csv;
+   VARIABLE:
+     NAMES ARE rating complaints privileges learning raises critical advance;
+   MODEL:
+     rating ON complaints privileges learning raises critical advance;
+ "
> write(myscript, "attitude.inp")
> list.files()

 [1] "attitude.csv"                    "attitude.inp"
 [3] "class4.Rnw"                      "class4.tex"
 [5] "desktop.ini"                     "IBM SPSS Statistics 20.lnk"
 [7] "Mplus Editor.lnk"                "R x64 3.1.0.lnk"
 [9] "RStudio.lnk"                     "SAS 9.4 (English).lnk"
[11] "StatTransfer Twelve (64 Bit).lnk" "Syntax N 1000"
```

*Mplus* can be run by the GUI program or by the command prompt (DOS in Windows or Linux Terminal in Macs). R allows us to control the command prompt by the `shell` function. Therefore, we can run *Mplus* via R.

```
> shell("mplus attitude.inp attitude.out")
> list.files()
```

```
 [1] "attitude.csv"                      "attitude.inp"
 [3] "attitude.out"                      "class4.Rnw"
 [5] "class4.tex"                        "desktop.ini"
 [7] "IBM SPSS Statistics 20.lnk"        "Mplus Editor.lnk"
 [9] "R x64 3.1.0.lnk"                   "RStudio.lnk"
[11] "SAS 9.4 (English).lnk"             "StatTransfer Twelve (64 Bit).lnk"
[13] "Syntax N 1000"
```

`"attitude.inp"` is used as an input and `"attitude.out"` is used as an output of *Mplus*. The output file should be in our current directory after running *Mplus*. Next, we can use the `readLines` function to read the output file into R console.

```
> output <- readLines("attitude.out")
```

The `output` object is a vector that different elements represent different lines of the *Mplus* output. Let's show the output in Lines 106–123. The results from *Mplus* are exactly the same as the results from the `lm` function.

```
> output[106:123]
```

```
 [1] ""
 [2] ""
 [3] "MODEL RESULTS"
 [4] ""
 [5] "                                                    Two-Tailed"
 [6] "                    Estimate       S.E.  Est./S.E.    P-Value"
 [7] ""
 [8] " RATING   ON"
 [9] "    COMPLAINTS         0.613      0.141      4.350      0.000"
[10] "    PRIVILEGES        -0.073      0.119     -0.615      0.539"
[11] "    LEARNING           0.320      0.148      2.171      0.030"
[12] "    RAISES             0.082      0.194      0.421      0.673"
[13] "    CRITICAL           0.038      0.129      0.298      0.766"
[14] "    ADVANCE           -0.217      0.156     -1.391      0.164"
[15] ""
[16] " Intercepts"
[17] "    RATING            10.787     10.147      1.063      0.288"
[18] ""
```

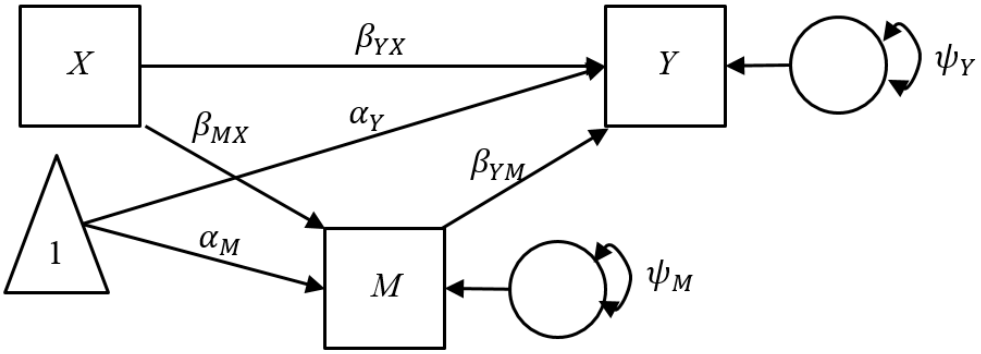
When we have the output in R console, we might want to delete some files in our current directory by the `file.remove` function.

```
> files <- c("attitude.csv", "attitude.inp", "attitude.out")
> file.remove(files)
```

```
[1] TRUE TRUE TRUE
```

Let's run a simulation study on mediation model using *Mplus*. Mediation model is used to find the mechanism behind the relationship between an independent variable, $X$, and a dependent variable, $Y$, via the third variable called mediator, $M$. That is, an independent variable influences a mediator and the mediator influences a dependent variable successively. The causal path that is explained by the mediator is called indirect effect. The causal path that the independent variable directly influences the dependent variable is called direct effect. The simplest mediation model involving with three variables, $X$, $Y$, and $M$, is as follows:

$$\begin{aligned} \hat{M} &= \alpha_M + \beta_{MX} X \\ \hat{Y} &= \alpha_Y + \beta_{YM} M + \beta_{YX} X \end{aligned} \tag{1}$$




The direct effect is $\beta_{YX}$, the indirect effect is $\beta_{YM} \times \beta_{MX}$, and the total effect is the sum of direct and indirect effects. Although many packages in R can run the mediation model, we will use *Mplus* here to illustrate the communication between R and other programs. Let's generate a single data set and fit the data set using *Mplus*.

```
> set.seed(123321)
> x <- rnorm(200, 0, 1)
> m <- 0.2 * x + rnorm(200, 0, sqrt(0.96))
> y <- 0.2 * x + 0.2 * m + rnorm(200, 0, sqrt(0.88))
> dat <- data.frame(x, m, y)
> write.table(dat, file = "example.csv", row.names = FALSE,
+     col.names = FALSE, sep = ",")
> exscript <- "
+   TITLE: Example on Mediation Model;
+   DATA:
```

```
+     FILE IS example.csv;
+   VARIABLE:
+     NAMES ARE x m y;
+   MODEL:
+     y ON x (c)
+          m (b);
+     m ON x (a);
+   MODEL CONSTRAINT:
+     new(ab);
+     ab = a * b;
+ "
> write(exscript, "example.inp")
> shell("mplus example.inp example.out")
> exout <- readLines("example.out")
> exout[110:131]

 [1] "MODEL RESULTS"
 [2] ""
 [3] "                                                    Two-Tailed"
 [4] "                    Estimate       S.E.  Est./S.E.    P-Value"
 [5] ""
 [6] " Y        ON"
 [7] "    X                 0.244      0.068      3.569      0.000"
 [8] "    M                 0.140      0.078      1.793      0.073"
 [9] ""
[10] " M        ON"
[11] "    X                 0.165      0.061      2.721      0.006"
[12] ""
[13] " Intercepts"
[14] "    M                 0.066      0.064      1.039      0.299"
[15] "    Y                -0.047      0.071     -0.657      0.511"
[16] ""
[17] " Residual Variances"
[18] "    M                 0.809      0.081     10.000      0.000"
[19] "    Y                 0.993      0.099     10.001      0.000"
[20] ""
[21] "New/Additional Parameters"
[22] "    AB                0.023      0.015      1.497      0.134"
```

In a simulation study, we would not like to use all information from the output. We need only specific outputs from the output text file. First, we need to find the line that has our desired outputs. In this case, the line containing the indirect effect estimate is desired. Thus, we need to find the line with `"AB"`. The `grep` function can be used to find the lines with the desired text.

```
> grep("AB", exout)
```

```
[1]  11 131
```

We have two lines containing the desired text: 11 and 131. Line 11 has `AB` in the word, `VARIABLE`. Line 131 is our desired line. Thus, we will extract Line 131 by the following script:

```
> exline <- exout[grep("AB", exout)[2]]
> exline

[1] "    AB                  0.023      0.015      1.497      0.134"
```

Next, we need to separate a single text line into multiple pieces based on the white spaces. The `strsplit` function can be used to separate it.

```
> exsplit <- strsplit(exline, " ")
> exsplit

[[1]]
 [1] ""      ""      ""      ""      "AB"    ""      ""      ""      ""      ""      ""
[12] ""      ""      ""      ""      ""      ""      ""      ""      ""      ""      "0.
[23] ""      ""      ""      ""      ""      "0.015" ""      ""      ""      ""      ""
[34] "1.497" ""      ""      ""      ""      ""      "0.134"
```

`exsplit` is in the list format with the length of 1. We will use the first element in the list. You will see that, in the first element, there are many `""`. We can remove it by using the `setdiff` function:

```
> exresult <- setdiff(exsplit[[1]], "")
> exresult

[1] "AB"    "0.023" "0.015" "1.497" "0.134"
```

Then, we can turn the last four numbers into the numeric format:

```
> as.numeric(exresult[2:5])

[1] 0.023 0.015 1.497 0.134
```

These four numbers are parameter estimate, standard error, $z$ test, and $p$ value of the indirect effect. In addition, to ensure that the analysis is convergent, we need to check whether the *Mplus* output has the following line: `THE MODEL ESTIMATION TERMINATED NORMALLY`. We can check it by the `grep` function as well.

```
> grep("THE MODEL ESTIMATION TERMINATED NORMALLY", exout)

[1] 61
```

If the results are numbers, this sentence is printed in the output so the result is convergent. However, if the results do not have any numbers (e.g., `integer(0)`), then this sentence is not printed so the result is not convergent. Anyway, let's remove all created files by the `file.remove` function.

```
> files <- c("example.csv", "example.inp", "example.out")
> file.remove(files)

[1] TRUE TRUE TRUE

> list.files()

 [1] "class4.Rnw"                          "class4.tex"
 [3] "desktop.ini"                         "IBM SPSS Statistics 20.lnk"
 [5] "Mplus Editor.lnk"                    "R x64 3.1.0.lnk"
 [7] "RStudio.lnk"                         "SAS 9.4 (English).lnk"
 [9] "StatTransfer Twelve (64 Bit).lnk" "Syntax N 1000"
```

Before we talk about the simulation study, let me introduce you the `paste` and `paste0` functions. Both functions are used to concatenate two or more texts together:

```
> paste("a", "b", "c")

[1] "a b c"

> paste("a", "b", "c", sep = "")

[1] "abc"

> paste0("a", "b", "c")

[1] "abc"
```

The `paste` function, by default, will concatenate texts into a single text with spaces that separate between the original texts. The spaces are eliminated if the `sep` argument is set as `""`. The `paste0` function is basically the `paste` function without spaces. We may use numbers instead of texts in the `paste` or `paste0` functions:

```
> paste0("script", 2, ".inp")

[1] "script2.inp"
```

Now, let's run a Monte Carlo simulation on the mediation model. The only design condition in this study is sample size: 50, 100, 200, 400, and 800. The number of replications is 1,000. The steps of running this simulation are (a) creating folders for different sample sizes, (b) writing data files in each folder (1,000 each), (c) writing input files in each folder (1,000 each), (d) running all input files in all folders using *Mplus*, and (e) extracting the indirect effect estimates from all outputs from all folders.

The first step is to create folders for each sample size condition. Folders can be created by using the `dir.create` function:

```
> n <- c(50, 100, 200, 400, 800)
> for(i in 1:length(n)) dir.create(paste0("n", n[i]))
> list.files()

 [1] "class4.Rnw"                          "class4.tex"
 [3] "desktop.ini"                         "IBM SPSS Statistics 20.lnk"
 [5] "Mplus Editor.lnk"                    "n100"
 [7] "n200"                                "n400"
 [9] "n50"                                 "n800"
[11] "R x64 3.1.0.lnk"                     "RStudio.lnk"
[13] "SAS 9.4 (English).lnk"               "StatTransfer Twelve (64 Bit).lnk"
[15] "Syntax N 1000"
```

Folders `n50`, `n100`, `n200`, `n400`, and `n800` are created. Then, we need to generate data sets and save them in each folder. Nested for loop is used. The first loop goes over sample size conditions. The second for loop goes over replications.

```
> nRep <- 1000
> currentDir <- getwd()
> set.seed(123321)
> for(i in 1:length(n)) {
+   for(j in 1:nRep) {
+     x <- rnorm(n[i], 0, 1)
+     m <- 0.2 * x + rnorm(n[i], 0, sqrt(0.96))
+     y <- 0.2 * x + 0.2 * m + rnorm(n[i], 0, sqrt(0.88))
+     dat <- data.frame(x, m, y)
+     filename <- paste0("dat", j, ".csv")
+     targetFile <- paste0(currentDir, "/n", n[i], "/", filename)
+     write.table(dat, file = targetFile, row.names = FALSE,
+                 col.names = FALSE, sep = ",")
+   }
+ }
```

`nRep` represents the number of replications. `currentDir` saves the current directory. Inside the nested for loop, data are created with the sample size specified by `n[i]`. The datafile name has the number of replication tag (`j`). `targetFile` represents the location of data to be saved in the hard drive. The current directory is concatenated with an appropriate folder name and followed by the datafile name. `write.table` is used to save the data in R to a file at the specified location.

Next, let's create *Mplus* script. The strategy is to create a template script first. Then, an appropriate datafile name is substituted for an appropriate position in the template. The modified template will be saved as an *Mplus* input file into an appropriate directory.

```
> template <- "
+   TITLE:
+     My Simulation;
```

```
+   DATA:
+     FILE IS subFile;
+   VARIABLE:
+     NAMES ARE x m y;
+   MODEL:
+     y ON x (c)
+          m (b);
+     m ON x (a);
+   MODEL CONSTRAINT:
+     new(ab);
+     ab = a * b;
+ "
> for(i in 1:length(n)) {
+   for(j in 1:nRep) {
+     datname <- paste0("dat", j, ".csv")
+     tempscript <- gsub("subFile", datname, template)
+     filename <- paste0("script", j, ".inp")
+     targetFile <- paste0(currentDir, "/n", n[i], "/", filename)
+     write(tempscript, targetFile)
+   }
+ }
```

`template` contains the *Mplus* script to run a mediation model. Please notice at the file name. I used `subFile` to represent the place that I will substitute it by an appropriate file name later. Inside the nested for loop, `datname` is the datafile name given the number of replications, `j`. `datname` substitute `"subFile"` in the `template`. The output of the substitution is called `tempscript`. Next, the input file name is created in `filename`. The target location of the input file name is also created in `targetFile`. Finally, the input is written in the appropriate location using the `write` function.

Currently, we have input files and data files in each folder. This is the time to run *Mplus* scripts in all folders.

```
> for(i in 1:length(n)) {
+   targetDir <- paste0(currentDir, "/n", n[i], "/")
+   setwd(targetDir)
+   for(j in 1:nRep) {
+     filename <- paste0("script", j, ".inp")
+     outname <- paste0("output", j, ".out")
+     shell(paste0("mplus ", filename, " ", outname))
+   }
+ }
> setwd(currentDir)
```

We still use the nested for loop. The difference in this nested for loop and the previous nested for loops is that the working directory is changed to each sample size folder using

`setwd` function. After the working directory is set, the input file and output file for each replication are named and put in the `shell` function to run *Mplus*. Do not forget to change the working directory back to the original working directory.

Finally, let's extract the estimates of indirect effects from all output files:

```
> result <- NULL
> for(i in 1:length(n)) {
+   for(j in 1:nRep) {
+     targetFile <- paste0(currentDir, "/n", n[i], "/", "output", j, ".out")
+     tempoutput <- readLines(targetFile)
+     if(length(grep("THE MODEL ESTIMATION TERMINATED NORMALLY",
+             tempoutput)) > 0) {
+       targetline <- grep("AB", tempoutput)[2]
+       textline <- tempoutput[targetline]
+       textsplit <- strsplit(textline, " ")
+       textresult <- setdiff(textsplit[[1]], "")
+       result <- rbind(result, c(n[i], as.numeric(textresult[2:5])))
+     } else {
+           result <- rbind(result, c(n[i], rep(NA, 4)))
+         }
+   }
+ }
> colnames(result) <- c("n", "est", "se", "z", "p")
> aggregate(est ~ n, data = result, FUN = mean)
```

```
    n      est
1  50 0.040019
2 100 0.040915
3 200 0.039254
4 400 0.039871
5 800 0.039768
```

Inside the nested for loop, `targetFile` is the location of the output of the `j`-th replication in the `n[i]` folder. `tempoutput` extracts the output from the file. The if statement is used to check whether the file is convergent. If so, the results of the indirect effect are extracted from the appropriate line. The results of the indirect effect are attached to the `result` object using the `rbind` function. This process is repeated for all sample size conditions and all replications. After that, the `result` object has information from all output files. The `aggregate` function is used to find the average of parameter estimates for each sample size condition.

# 1 Exercise

Use the following code to run a simulation on the multiple-mediation model where the only design condition is sample size: 50, 250, and 1,000. Find the average of parameter estimates of total indirect effect (`indirect`):

```
> set.seed(123321)
> x <- rnorm(250, 0, 1)
> m1 <- 0.2 * x + rnorm(250, 0, sqrt(0.96))
> m2 <- 0.2 * x + rnorm(250, 0, sqrt(0.96))
> y <- 0.2 * x + 0.2 * m1 + 0.2 * m2 + rnorm(250, 0, sqrt(0.84))
> dat <- data.frame(x, m1, m2, y)
> write.table(dat, file = "exercise.csv", row.names = FALSE,
+     col.names = FALSE, sep = ",")
> exercisescript <- "
+   TITLE: Example on Mutliple-Mediation Model;
+   DATA:
+     FILE IS exercise.csv;
+   VARIABLE:
+     NAMES ARE x m1 m2 y;
+   MODEL:
+     y ON x (c)
+          m1 (b1)
+                  m2 (b2);
+     m1 ON x (a1);
+         m2 ON x (a2);
+   MODEL CONSTRAINT:
+     new(ab1, ab2, indirect);
+     ab1 = a1 * b1;
+         ab2 = a2 * b2;
+         indirect = ab1 + ab2;
+ "
> write(exercisescript, "exercise.inp")
> shell("mplus exercise.inp exercise.out")
> exerciseout <- readLines("exercise.out")
> exerciseout[140:143]

[1] "New/Additional Parameters"
[2] "    AB1                0.024      0.013      1.840      0.066"
[3] "    AB2                0.035      0.016      2.254      0.024"
[4] "    INDIRECT           0.059      0.020      2.891      0.004"
```