**Functionsin R Script to be Called**

R scripts for Lasso estimation is written as a separate file. In particular, glmnet package is used. Like this, when using another package library, one more rJava command is also needed, which is discussed later.

* shlee\_RLib.R

|  |  |  |
| --- | --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | library(glmnet)    test\_glmnet <– function(nvar) {        # artificial data      x = matrix(rnorm(100\*nvar), 100, nvar)      y = rnorm(100)        # Lasso estimation      fit1 = glmnet(x, y, alpha = 1)        # coefficient matrix with lambda = 0.01, 0.05      sm.coef <– as.matrix(unlist(coef(fit1, s=c(0.01,0.05))))        return(as.matrix(sm.coef))  }    [*Colored by Color Scripter*](http://colorscripter.com/info#e) | [cs](http://colorscripter.com/info#e) |

With source() command, test\_glmnet() function in shlee\_RLib.R can be called in another R script as follows.

|  |  |  |
| --- | --- | --- |
| 1  2  3  4  5  6 | rm(list = ls()) # remove all files from your workspace    source(“D:/SHLEE/rJava/code/shlee\_RLib.R”)    test\_glmnet(10)    [*Colored by Color Scripter*](http://colorscripter.com/info#e) | [cs](http://colorscripter.com/info#e) |

test\_glmnet(10) command returns the folloiwng results, which are coefficient vectors of two Lasso models (\(\lambda = 0.01, 0.05\)).

|  |  |  |
| --- | --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | > test\_glmnet(10)                        1            2  (Intercept) –0.16453256 –0.159861599  V1          –0.22338346 –0.179309971  V2          –0.01422389  0.000000000  V3           0.03766415  0.000000000  V4           0.03100075  0.000000000  V5          –0.03618045 –0.003826916  V6           0.06392980  0.021847332  V7           0.05765196  0.022836526  V8           0.02248489  0.000000000  V9          –0.09365914 –0.046769476  V10         –0.19398003 –0.180170304  >  [*Colored by Color Scripter*](http://colorscripter.com/info#e) | [cs](http://colorscripter.com/info#e) |

**Calling User-defined Functions in Eclipse Java**

At first, let’s make an Eclipse java class file in which R functions are called using rJava. We name it CRJava2.class for example. In CRjava2.class file, write the folloiwng java code.

|  |  |  |
| --- | --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70 | //=========================================================================#  //————————————————————————-#  //rJava example with user-defined function which reduces so much lines.  //=========================================================================#  package aRjava;    import org.rosuda.JRI.Rengine;  import org.rosuda.JRI.REXP;    // Run Config -> Environment 3 copy and paste  // That’s all there is to it and nothing else is needed.    public class CRjava2 {      public static void main(String[] args) {            // Launch and Start rJava          Rengine re=new Rengine(new String[] { “–vanilla” }, false, null);            //——————————————————————          // Very Important !!!!!!!!!!!!!!!!!!          //——————————————————————          // User-defined function use glmnet library.          //          // Without this command, Java produces the following error.          // :: Error in library(glmnet) :          //    there is no package called ‘glmnet’          //          // To sidestep this error, following command is recommended.          //          // glmnet package is located          // at C:/Users/shlee/Documents/R/win-library/4.0          //——————————————————————          re.eval(“.libPaths(‘C:/Users/shlee/Documents/R/win-library/4.0’)”);            // R file with its local directory          // in which user-defined functions are written          re.eval(“source(‘D:/SHLEE/rJava/code/shlee\_rlib.R’)”);            // Input parameters          int nvar = 5; int ncol = 2;            // Call user-defined R function          REXP x = re.eval(“test\_glmnet(“+nvar+“)”);            // 1) Result : raw output          System.out.println(“1) REXP result : raw output”);          System.out.println(x);            // 2) Results : rounded output          System.out.println(“\n2) REXP result : formatted output using 2D array”);            // R matrix t –> Java 2D array          double[][] mout = x.asDoubleMatrix();            for(int i = 0; i<nvar; i++) {              for(int j = 0; j<ncol; j++) {                  System.out.print(String.format(“%2.5f”, mout[i][j]) + ”   “);              }              System.out.println(“”);          }            // end rJava          re.end();      }  }    [*Colored by Color Scripter*](http://colorscripter.com/info#e) | [cs](http://colorscripter.com/info#e) |

In fact, it is interesting that the essencial part of the above R jave code is calling test\_glmnet(). We can save many lines of code, which depends on the extent or size of calcuations or estimations.

|  |  |  |
| --- | --- | --- |
| 1  2 | REXP x = re.eval(“test\_glmnet(“+nvar+“)”);    [*Colored by Color Scripter*](http://colorscripter.com/info#e) | [cs](http://colorscripter.com/info#e) |

**Point to Note**

There is one important thing to know. When other (not built-in) libraries such as glmnet are included, the following rJava command is necessary. It is important.

|  |  |  |
| --- | --- | --- |
| 1  2 | re.eval(“.libPaths(‘C:/Users/shlee/Documents/R/win-library/4.0’)”); | [cs](http://colorscripter.com/info#e) |

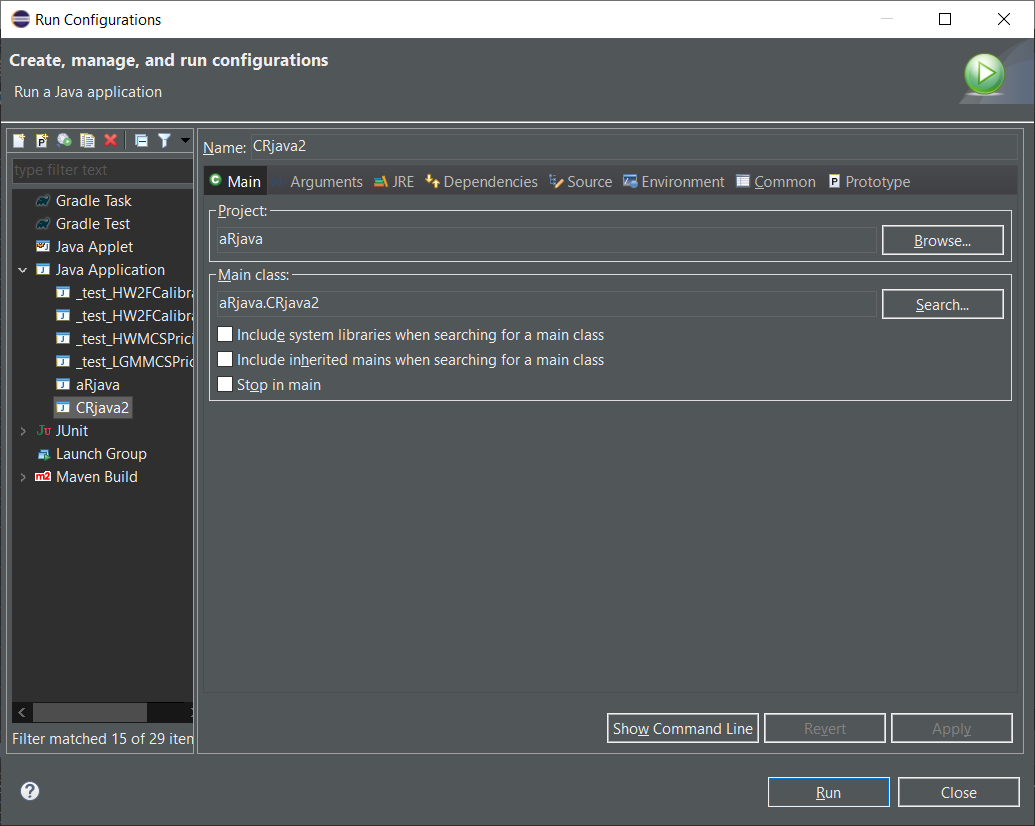
Without the above rJava command, Eclipse returns an error message with “there is no package called ‘glmnet'”.

**First Run and Errors**

When we run the above Java code, we encounter the following errors. Hence, we need to do some settings for CRjava2.class file.

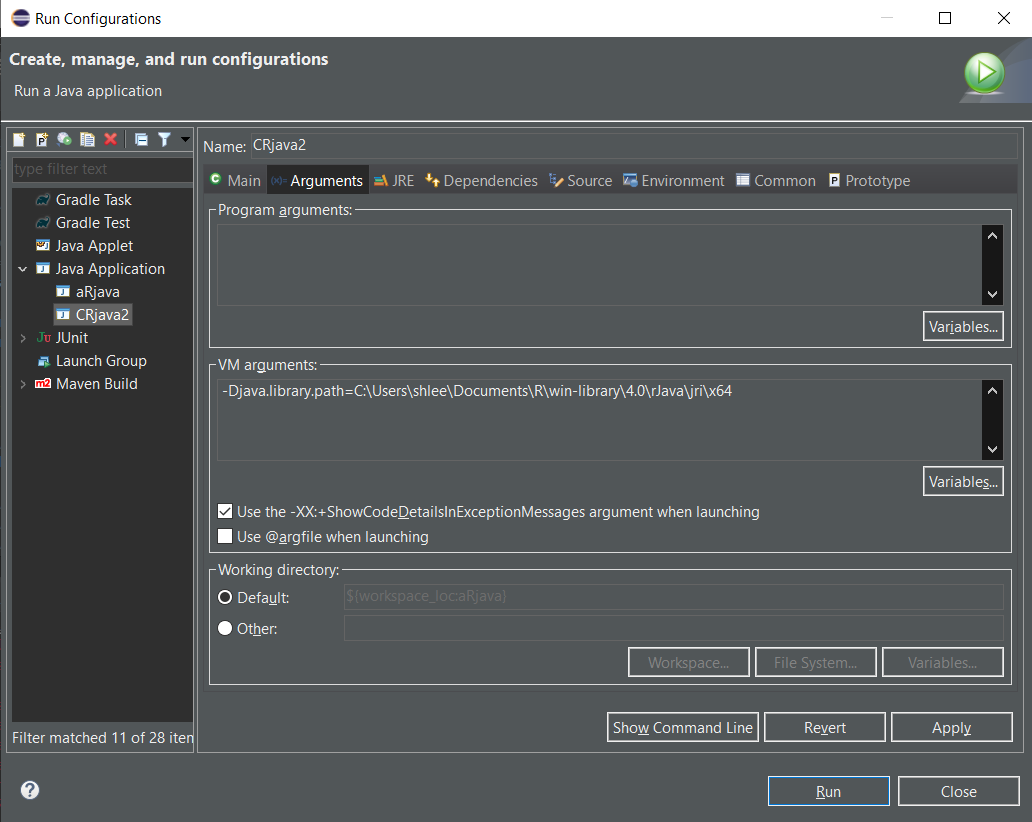
But this first running this project is important because after this trial, **Run Configuration** (which will be explained later) can identify this project. 

**Setting for New Class File**

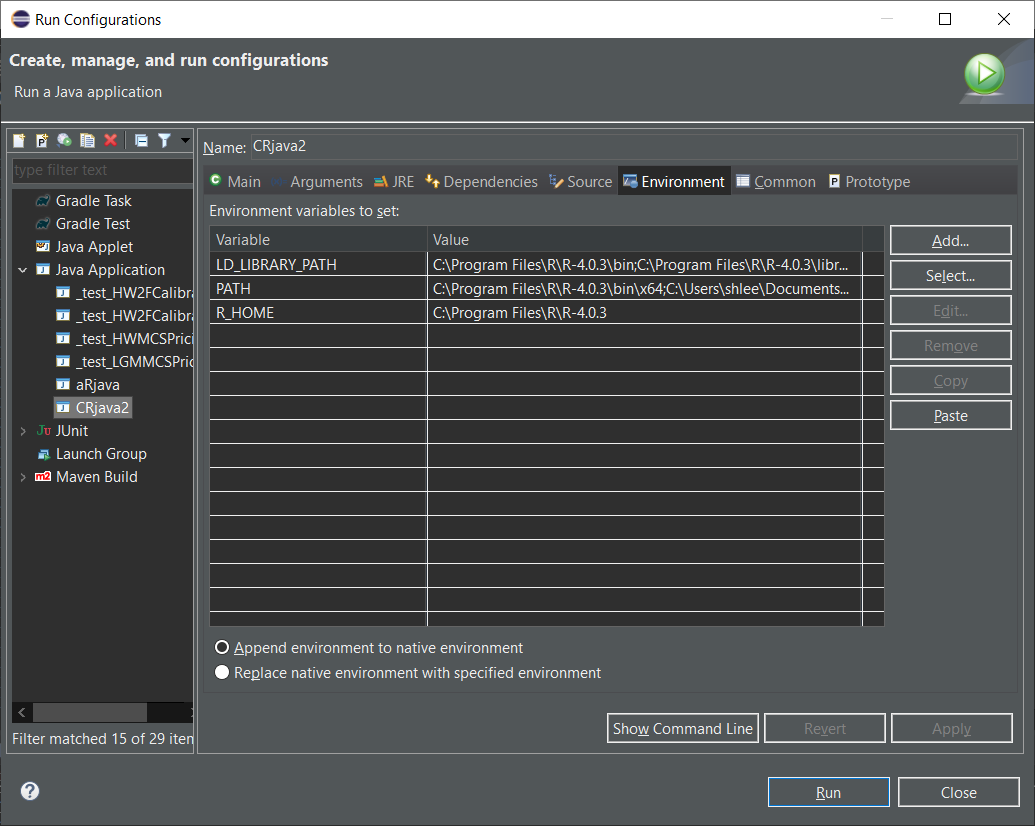
Two settings on CRjava2.class file are necessary. After right mouse clicking on CRjava2.class file, select Run As –> Run Configurations. 

In **Arguments** tab, **VM arguments** is filled as follows (Use copy and paste from aRjava setting).

* VM arguments : -Djava.library.path=C:\Users\shlee\Documents\R\win-library\4.0\rJava\jri\x64

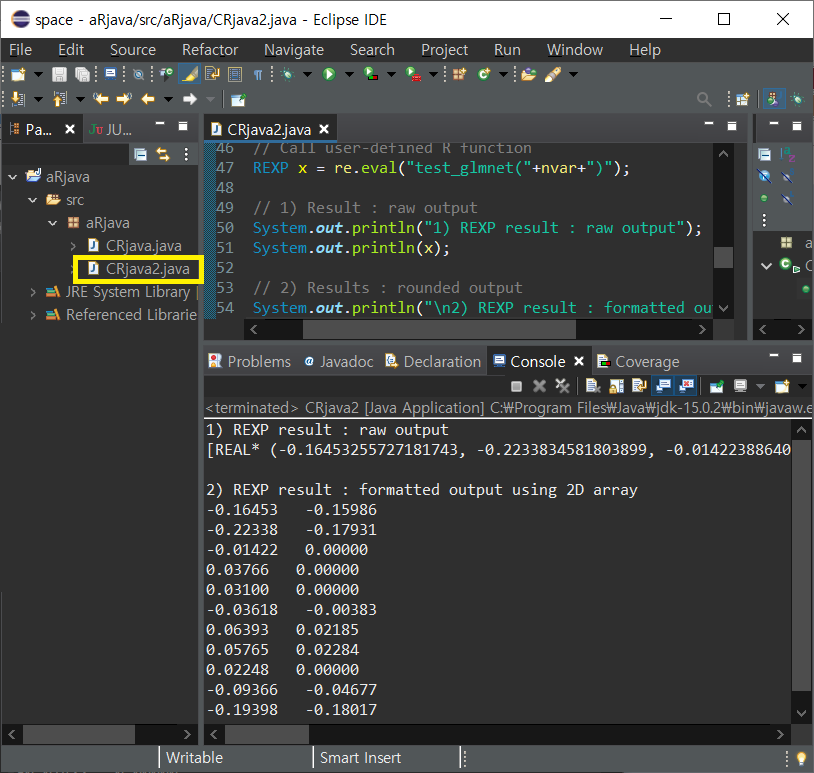
  
In **Environment** tab, Add three directories in the following way (Use copy and paste from aRjava setting with buttons).

* LD\_LIBRARY\_PATH : C:\Program Files\R\R-4.0.3\bin;C:\Program Files\R\R-4.0.3\library;C:\Users\shlee\Documents\R\win-library;
* PATH : C:\Program Files\R\R-4.0.3\bin\x64;C:\Users\shlee\Documents\R\win-library\rJava\jri\x64;
* R\_HOME : C:\Program Files\R\R-4.0.3



Now the setting for the added class file is done completely.

**Running and Results**



When we rerun CRjava2.class file, We can obtain correct results.

We can find that results from only R and Eclipse with rJava are same.   
  
Having done the overall environment setting already, we have only to add two settings on new added class file simply.   
  
From this post, we can make Java code with rJava compact from calling user-defined R functions efficiently. This will help reduce many lines to essencially one line and enhance code readability. \(\blacksquare\)