Topic 2: Exercise 1

Daniel Alonso

November 28th, 2020

Importing libraries

```r
library(dplyr)
library(Rcpp)
library(JuliaCall)
```

Importing data as described by exercise

```r
d <- read.csv("../../datasets/Colleges.csv")
```

Replacing binary variable Private with 1 and 0

```r
d$Private <- ifelse(d$Private == "Yes", 1, 0)
```

Selecting columns

```r
data <- d %>% dplyr::select('Private','Apps','Accept','Enroll','F.Undergrad')
```

Calculating covariances

```r
cov_matrix <- cov(data)
cov_matrix
#>                  Private          Apps        Accept       Enroll  F.Undergrad
#> Private        0.1986559     -745.3552     -519.2042    -235.1942    -1330.764
#> Apps        -745.3552439 14978459.5301  8949859.8119 3045255.9876 15289702.474
#> Accept      -519.2042169  8949859.8119  6007959.6988 2076267.7627 10393582.435
#> Enroll      -235.1942393  3045255.9876  2076267.7627  863368.3923  4347529.884
#> F.Undergrad -1330.7637175 15289702.4742 10393582.4355 4347529.8841 23526579.326
```

Calculating correlations

```
corr_matrix <- cov2cor(cov_matrix)
corr_matrix
#>                Private       Apps     Accept     Enroll F.Undergrad
#> Private      1.0000000 -0.4320947 -0.4752520 -0.5679078  -0.6155605
#> Apps        -0.4320947  1.0000000  0.9434506  0.8468221   0.8144906
#> Accept      -0.4752520  0.9434506  1.0000000  0.9116367   0.8742233
#> Enroll      -0.5679078  0.8468221  0.9116367  1.0000000   0.9646397
#> F.Undergrad -0.6155605  0.8144906  0.8742233  0.9646397   1.0000000
```

Experimenting a little bit with the private variable

Let's try changing the Yes to 0 and the No to 1 and checking the covariances and correlations

```
d <- read.csv("../../datasets/Colleges.csv")
d$Private <- ifelse(d$Private == "Yes", 0, 1)
data <- d %>% dplyr::select('Private','Apps','Accept','Enroll','F.Undergrad')
```

```
cov_matrix <- cov(data)
cov_matrix
#>                  Private         Apps       Accept       Enroll  F.Undergrad
#> Private        0.1986559 7.453552e+02 5.192042e+02     235.1942     1330.764
#> Apps         745.3552439 1.497846e+07 8.949860e+06 3045255.9876 15289702.474
#> Accept       519.2042169 8.949860e+06 6.007960e+06 2076267.7627 10393582.435
#> Enroll       235.1942393 3.045256e+06 2.076268e+06  863368.3923  4347529.884
#> F.Undergrad 1330.7637175 1.528970e+07 1.039358e+07 4347529.8841 23526579.326
corr_matrix <- cov2cor(cov_matrix)
corr_matrix
#>               Private      Apps    Accept    Enroll F.Undergrad
#> Private     1.0000000 0.4320947 0.4752520 0.5679078   0.6155605
#> Apps        0.4320947 1.0000000 0.9434506 0.8468221   0.8144906
#> Accept      0.4752520 0.9434506 1.0000000 0.9116367   0.8742233
#> Enroll      0.5679078 0.8468221 0.9116367 1.0000000   0.9646397
#> F.Undergrad 0.6155605 0.8144906 0.8742233 0.9646397   1.0000000
```

We get the same numbers with reversed signs.

We define the following function (in julia) to help our understanding:

Takes the arguments:

```
- nrows: number of data to simulate (amount of rows)
- simulations: number of different times to simulate and average the results
- fixed_value_col: boolean parameter with true -> assigns a set of values between mins[1] and maxs[1] (
- reverse: boolean parameter that determines whether the 0s in the binary variable are assigned to the l
- sim_binaries: this would simulate a rolling proportion of binaries where iteration 1 has all zeros in
- mins: array containing as first element the minimum value to use for corresponding values in the quan
- maxs: same as minimum but they're the maximums.
```

Example of how the dataset changes for a run of the function with parameters: nrows=5, simulations=1 and the rest of the parameters as default:

```
[1.0 6565.0 6565.0; 0.0 7.0 8180.0; 0.0 1.0 6274.0; 0.0 5.0 4544.0; 0.0 8.0 3441.0]
[1.0 6565.0 6565.0; 1.0 8180.0 8180.0; 0.0 1.0 6274.0; 0.0 5.0 4544.0; 0.0 8.0 3441.0]
[1.0 6565.0 6565.0; 1.0 8180.0 8180.0; 1.0 6274.0 6274.0; 0.0 5.0 4544.0; 0.0 8.0 3441.0]
[1.0 6565.0 6565.0; 1.0 8180.0 8180.0; 1.0 6274.0 6274.0; 1.0 4544.0 4544.0; 0.0 8.0 3441.0]
[1.0 6565.0 6565.0; 1.0 8180.0 8180.0; 1.0 6274.0 6274.0; 1.0 4544.0 4544.0; 1.0 3441.0 3441.0]

julia>
```

Figure 1: Example of a simulated dataset with the function simulation_general in the Julia REPL

In Figure 1 we can see 5 iterations (as there's 5 simulated rows) using the function, where the leftmost value of each row is a binary variable (1 or 0), starting with (1,0,0,0,0) and ending with (1,1,1,1,1), and for our quantitative variable (with which we will calculate cov/corr) we can see the values go from a high value (copied from the 3rd column) and the rest of values being small (6565,7,1,5,8) and ending with large values (copied from column 3) being (6565, 8180, 6214, 4544, 3441).

Function definition

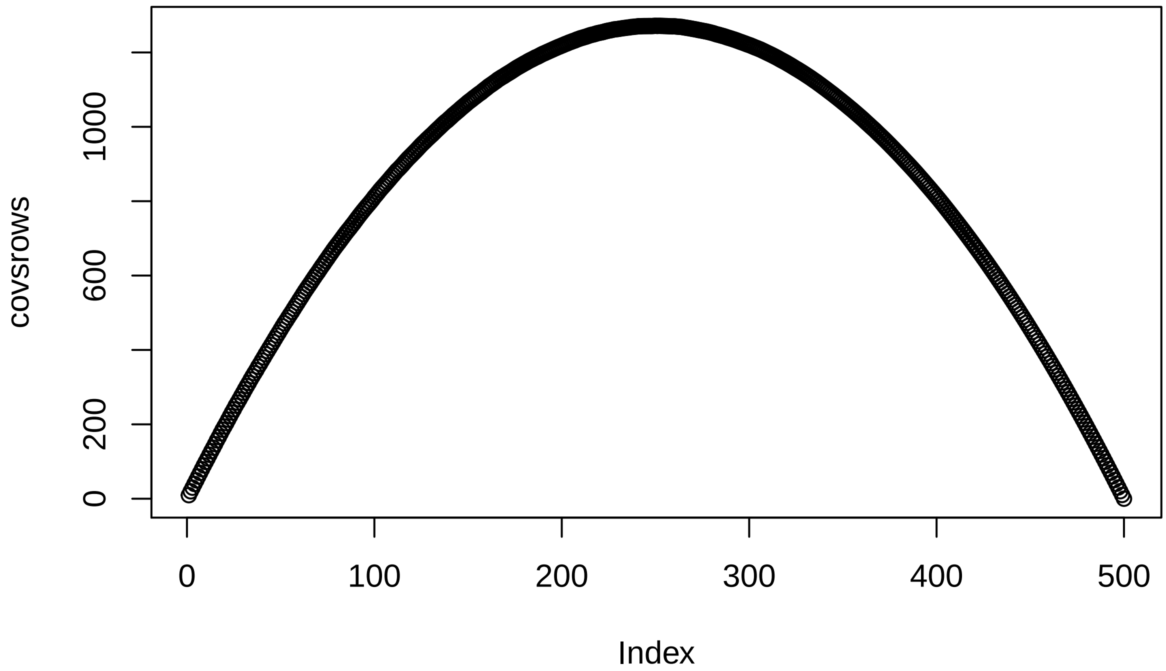
```julia
using Random
using Statistics
using Plots
gr()
#> Plots.GRBackend()

function simulation_general(nrows, simulations; fixed_value_col=false, reverse=false,
                             sim_binaries=true, mins=[1,100], maxs=[10,10000])
    # cov and corr matrixes
    covs = zeros(Float64, nrows, simulations)
    corr = zeros(Float64, nrows, simulations)

    # loop
    for s in 1:simulations
        pvtapps = zeros(Float64, nrows, 3)
        if sim_binaries == false
            pvtapps[:,1] = rand(0:1, nrows)
        end

        # random numbers column (quant variable)
        if fixed_value_col == true
            pvtapps[:,2] = rand(mins[1]:maxs[1],nrows)
        else
```

```
            if reverse == false
                pvtapps[:,2] = rand(mins[1]:maxs[1],nrows)
                pvtapps[:,3] = rand(mins[2]:maxs[2],nrows)
            else
                pvtapps[:,2] = rand(mins[2]:maxs[2],nrows)
                pvtapps[:,3] = rand(mins[1]:maxs[1],nrows)
            end
        end

        # loop for changing values
        for i in 1:nrows

            if sim_binaries == true
                pvtapps[1:i,1] = ones(i)
            end

            pvtapps[1:i,2] = pvtapps[1:i,3]

            # calculate corr and cov
            covs[i,s] = cov(pvtapps[:,1],pvtapps[:,2])
            corr[i,s] = cor(pvtapps[:,1],pvtapps[:,2])
        end
    end

    # results
    covsrows = zeros(Float64, nrows)
    corrrows = zeros(Float64, nrows)
    for i in 1:nrows
        covsrows[i] = mean(covs[i,:])
        corrrows[i] = mean(corr[i,:])
    end

    # return matrixes
    return covsrows, corrrows
end
#> simulation_general (generic function with 1 method)
```

```
covsrows, corrrows = simulation_general(500,200, reverse=false, sim_binaries=true);
```

Covariance plot with 500 data points and 100 simulations averaged

```
covsrows <- JuliaCall::julia_eval("covsrows")
plot(covsrows)
```



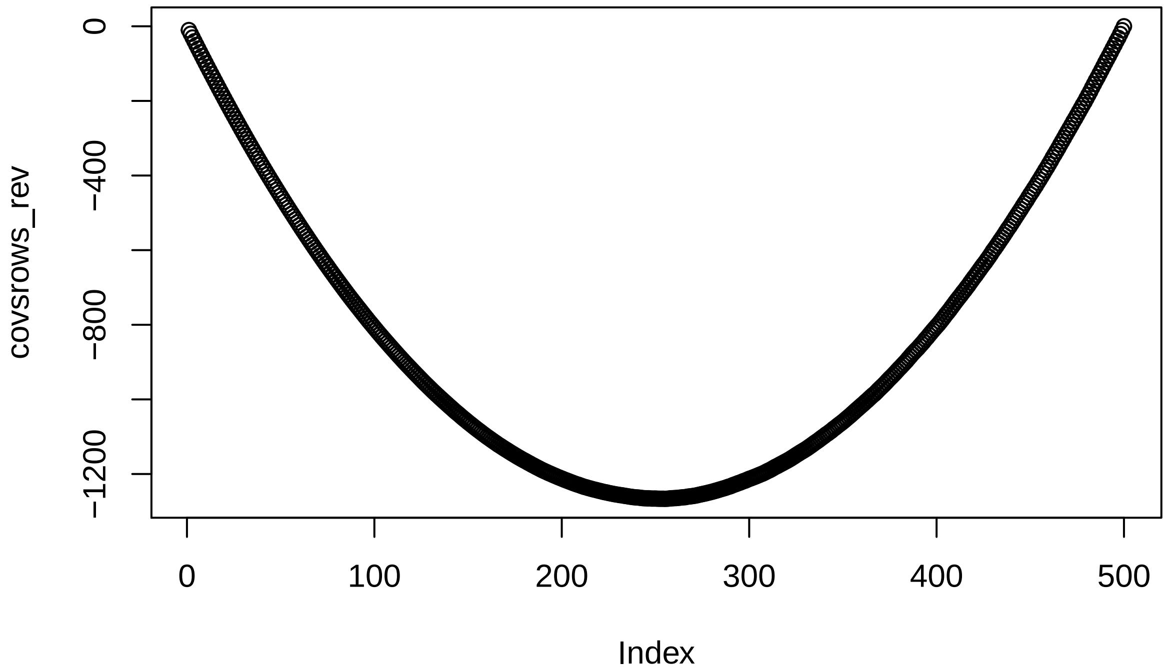
Each point here corresponds to one iteration of the function (leftmost being an iteration where the binary variable had its first value as 1 and the rest as 0 and rightmost being all 1s). For each 1 in the binary variable we have a value between 100 and 10000 in the quantitative variable used to calculate the covariance. For each 0 we have a value between 1 and 10, therefore, all values in the quantitative variable corresponding to a 0 in the binary are at least an order (x10) of magnitude larger than those corresponding to a 1.

Clearly, as long as values in the quantitative variable (corresponding to 1 in the binary variable) remain significantly larger than those corresponding to 0 the 0 in the binary variable, our covariance will grow as the proportion of 1s grow, however, once we reach half and half (half 0s and half 1s in the binary variable), our function reaches its global maximum and becomes a decreasing function.

The opposite thing would happen if we reverse the values, so then we would have the values corresponding to the binary variable's 1 to the smaller values (1-10) and larger values (100-10000) corresponding to the binary variable's 0.

```
covsrows_rev, corrrows_rev = simulation_general(500,200, reverse=true, sim_binaries=true);
```

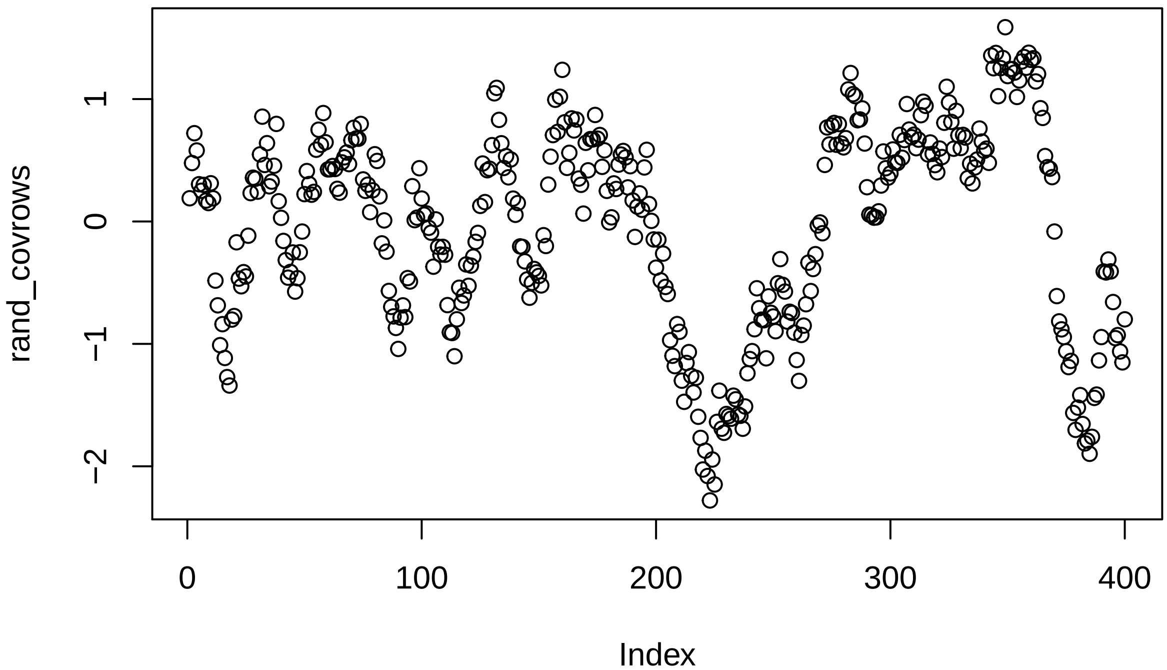
```
covsrows_rev <- JuliaCall::julia_eval("covsrows_rev")
plot(covsrows_rev)
```



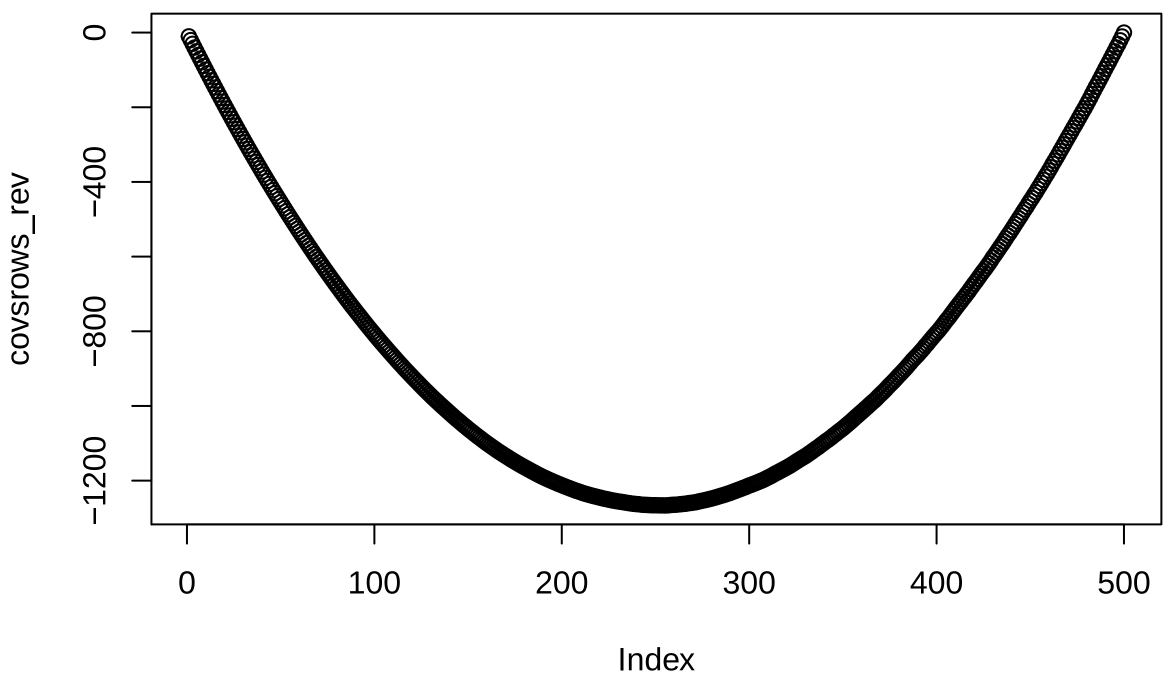
If we only simulate without considering the binary variable, so basically keeping a somewhat even amount of ones and zeros in it and randomizing the values of the quantitative variable, then we get no discernible pattern:

```
rand_covrows, rand_corrrows = simulation_general(400,1000,
                                        sim_binaries=false, fixed_value_col=false);
```

```
rand_covrows <- JuliaCall::julia_eval("rand_covrows")
plot(rand_covrows)
```

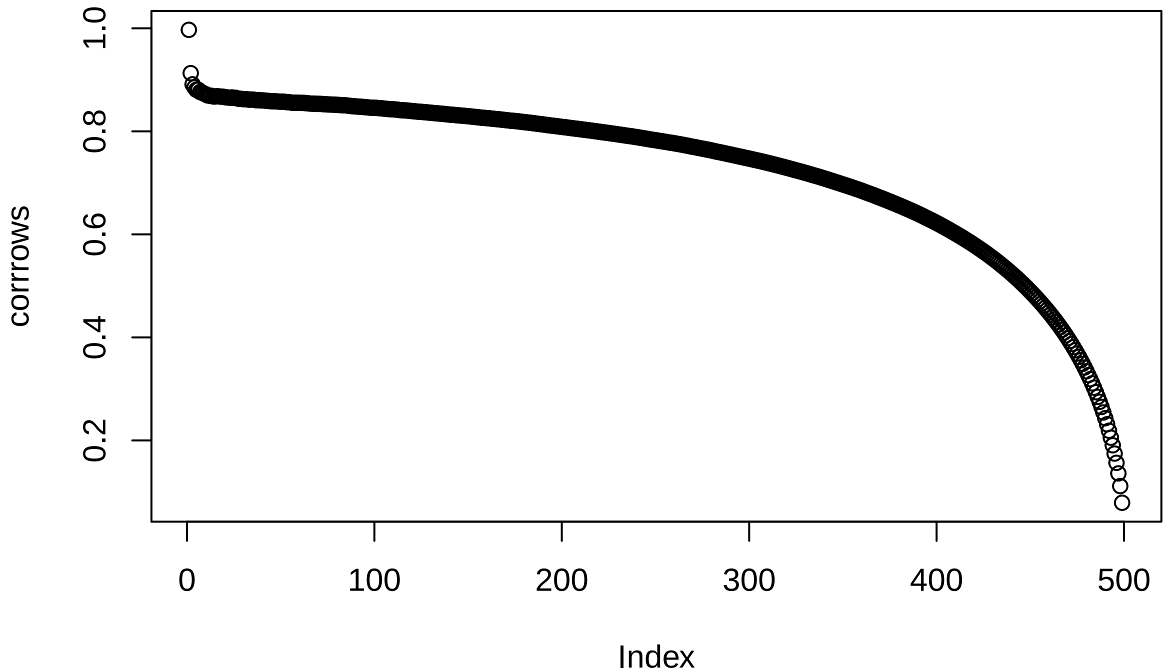



```r
covsrows_rev <- JuliaCall::julia_eval("covsrows_rev")
plot(covsrows_rev)
```



Correlation plot with 500 data points and 100 simulations averaged

```r
corrrows <- JuliaCall::julia_eval("corrrows")
plot(corrrows)
```



OBSERVATIONS

Now let's play around changing the size of the values that correspond in the quantitative variable to 1s or 0s in the binary column.

What information does the sample covariance provide?

We know that because the Private variable (binary variable) has only 2 possible values, its covariance with other variables is always going to be relatively small and will not provide much information.

What information does the sample correlation provide?