University of Duisburg-Essen
Faculty of Business Administration and Economics
Chair of Econometrics

A Functional Approach to (Parallelised) Monte Carlo Simulation

Advanced R for Econometricians

Final Project

Submitted to the Faculty of
Business Administration and Economics
at the
University of Duisburg-Essen

from:

Alexander Langnau, Öcal Kaptan, Sunyoung Ji

Matriculation Number:	232907, 230914, 229979
Study Path:	M.Sc. Econometircs
Reviewer:	Prof. Dr. Christoph Hanck
Secondary Reviewer:	M.Sc. Martin C. Arnold, M.Sc. Jens Klenke
Semester:	1^{st} Semester
Graduation (est.):	Summer Term 2022
Deadline:	09. 09. 2022

Contents

List of Tables

List of Figures

1 Introduction

Monte Carlo, named after a casino in Monaco, simulates complex probabilistic events using simple random events, such as the tossing of a pair of dice to simulate the casino's overall business model. In Monte Carlo computing, a pseudo-random number generator is repeatedly called which returns a real number in [0, 1], and the results are used to generate a distribution of samples that is a fair representation of the target probability distribution under study. (**Barbu**) Monte Carlo Method is combined with programming in modern research and contributes to various studies.

Monte Carlo simulations are and will stay an important method in the tool box of any econometrician, statistican or data scientist. Since these simulations may be needed on a regular basis or are run over a complex set of functions and parameters, its time well spend to implement some tools, that allow the user to easily create a variety of different Monte Carlo studies.

This paper was the final project of the course "Advanced R for econometricians" at the chair of econometrics at university Duisburg Essen. The goal is to use a functional programming aproach to create a collection of different wrapper functions in R, that - providing a convenient interface for Monte Carlo Simulations - create a paramter grid - iterate homogenous function calls over the parameter grid - provides an informative summary of the simulation results - can be visualized by ggplot-methods - offers the possibility to use parallelised processing (using `furrr` package)

A functional programming approach is well suited to implement the different steps. The structure of this paper underlying code in general follows this approach:

In chapter xyz we introduce different functions, that each specifically solve the task of the bullet points mentioned above. In the beginning we´ll underline the motivation and problem behind each function and showcase the code.

At the end of each section we provide a minimal working example, that illustrates the function and its output. We tried to implement in a way, that the function works for as much cases, as possible. If there are some restrictions regarding the usage of those functions, we´ll briefly discuss them as well.

2 Preprocess / Helper functions

2.1 Function for creating grid

In order to efficiently run a Monte Carlo simulation, we first need to specify for which set of parameters we want to run the simulation process. In case more than one variable is defined, it reasonable to create a parameter grid for each different combination of parameters.

```
create_grid <- function(parameters, nrep){
  input <- parameters
  storage <- list()
  name_vec <- c()
```

```r
  for(i in 1:length(input)){ #1:3
    a <- as.numeric(input[[i]][[2]])
    b <- as.numeric(input[[i]][[3]])
    c <- as.numeric(input[[i]][[4]])
    output <- seq(from=a, to=b, by=c)
    storage[[i]] <-  output
    name_vec[i] <- input[[i]][[1]]
  }

  grid <- expand_grid(unlist(storage[1])
                      , unlist(storage[2])
                      , unlist(storage[3])
                      , unlist(storage[4])
                      , unlist(storage[5])
                      , c(1:nrep))

  names(grid) <- c(name_vec, "rep")

  return(grid)
}
```

`create_grid` is the function hat creates a parameter grid with all permutations of the given parameters. The user has to input the parameters as a list, thats specified in the following way:

`parameter_list <- list(c("variable name 1", from, to, by) ,c("variable name 2", from, to, by)  ,c("variable name 3", from, to, by)  ,c("variable name 4", from, to, by))`

The function works with a minimum of 1 and a maximum of 4 variables. The structure of the remaining arguments is kept similar to the way the way R´s build in function`seq()` is specified: The first argument after the variable name defines the start of the sequence, the second one the end and the last one the steps, by which each variable specified for the parameter grid.

It would be fairly easy to adapt this helper function for more parameters, but it is assumed, that a parameter grid with up to 4 parameters offers enough complexity for the simulation. The function basically takes the infromation of the input parameter list and creates a parameter grid with `tidyr::expand_grid()`. The function also makes sure that the columns are named after the correct variable and also creates a different row for each number repetition, that the user specified in the second argument of `create_grid()`, namely `nrep`.

Following is a demonstration of how the input and output of this function looks like:

`create_grid()` Example:

```r
#four parameters
param_list0 <- list(c("n", 10, 20, 10)
                    ,c("mu", 0, 0.5, 0.25)
                    ,c("sd", 0, 0.3, 0.1)
                    ,c("gender", 0, 1, 1))

head(create_grid(param_list0, nrep=3), n=20)
```

```
## # A tibble: 20 x 5
##         n    mu    sd gender   rep
##     <dbl> <dbl> <dbl>  <dbl> <int>
##  1     10     0 0          0     1
##  2     10     0 0          0     2
##  3     10     0 0          0     3
##  4     10     0 0          1     1
##  5     10     0 0          1     2
##  6     10     0 0          1     3
##  7     10     0 0.1        0     1
##  8     10     0 0.1        0     2
##  9     10     0 0.1        0     3
## 10     10     0 0.1        1     1
## 11     10     0 0.1        1     2
## 12     10     0 0.1        1     3
## 13     10     0 0.2        0     1
## 14     10     0 0.2        0     2
## 15     10     0 0.2        0     3
## 16     10     0 0.2        1     1
## 17     10     0 0.2        1     2
## 18     10     0 0.2        1     3
## 19     10     0 0.3        0     1
## 20     10     0 0.3        0     2
```

2.2 Data generation function

Our second helper function called `data_generation()` uses the parameter grid (`grid`) and a user defined function for data generation (specified as input named `simulation`), f.e. `rnorm()`, `runif()` or `rpois()` to create data under the exact parameters specified in the grid. The draw of data points for each row of the parameter grid gets stored as a seperate element in a list.

Dependend on how many parameters were specified by the user, different functions of the `purrr`-package are are used, to run the specified function over the parameter grid. In case the user is looking to improve performance, a way of choosing a parallelised processing function of the `furrr`-package is offered.

- `map()` is used for one parameter and `future_map()` function is used for parallel process instead of `map()` function. Same for other `map()` functions as seen below.
- `map2()` is used for two parameters and `future_map2()` function is used for parallel process instead of `map2()` function.
- `pmap()` is used for three or more parameters and `future_pmap()` function is used for parallel process instead of `pmap()` function.

`options = furrr_options(seed = TRUE)` is used for reproducible random number generation process. This argument takes control of the RNG process for paralleization. It generates the same numbers for the given seed. More details can be found by running the command `?furrr_options` in RStudio.

```
data_generation <- function(simulation, grid){ #this is for use inside the function

  if(ncol(grid)==2){
    var1 <- c(unlist(grid[,1]))
    if(cores>1){
      data <- future_map(var1, simulation,.options = furrr_options(seed = TRUE))
    }else{
      data <- map(var1, simulation)
    }
  }

  if(ncol(grid)==3){
    var1 <- c(unlist(grid[,1]))
    var2 <- c(unlist(grid[,2]))
    if(cores>1){
      data <- future_map2(var1, var2, simulation,.options = furrr_options(seed = TRUE))
    } else{
      data <- map2(var1, var2, simulation)
    }
  }

  if(ncol(grid)==4){ #need to implement more than 3?!
    var1 <- c(unlist(grid[,1]))
    var2 <- c(unlist(grid[,2]))
    var3 <- c(unlist(grid[,3]))
    list1 <- list(var1,var2,var3)
    if(cores>1){
      data <- future_pmap(list1, .f=simulation,.options = furrr_options(seed = TRUE))
    }else{
      data <- pmap(list1, .f=simulation)
```

```
    }
  }

  return(data)
}
```

Following, we´d like to demonstrate the data genaration function with a simple example using the normal distribution (`rpois()`) as the underlying data generating process. At this point we waive running an example without parallel process of our data generating function, since the output wouldn't look different compared to the previous example. At a later point in the paper we´ll showcase the difference in computation time for the different functions.

`data_generation()` Example:

```
cores <- 1
#param_list1 <- list(c("n", 10, 20, 10))

param_list2 <- list(c("n", 10, 20, 10)
                  ,c("lambda", 0.5, 1, 0.5))
#create_grid(param_list1, nrep=10)
#create_grid(param_list1, nrep=1)

'grid1 <- create_grid(param_list1, nrep=3)
tail(data_generation(simulation=rnorm, grid=grid1),1)'
```

```
## [1] "grid1 <- create_grid(param_list1, nrep=3)\ntail(data_generation(simulation=rnorm, gr
```

```
grid2 <- create_grid(param_list2, nrep=1)
sim1 <- data_generation(simulation=rpois, grid=grid2)

grid2
```

```
## # A tibble: 4 x 3
##       n lambda   rep
##   <dbl>  <dbl> <int>
## 1    10    0.5     1
## 2    10    1       1
## 3    20    0.5     1
## 4    20    1       1
```

```
str(sim1)
```

```
## List of 4
##  $ n1: int [1:10] 0 1 0 1 2 0 0 1 0 0
##  $ n2: int [1:10] 3 1 1 1 0 2 0 0 0 3
##  $ n3: int [1:20] 1 1 1 3 1 1 0 0 0 0 ...
##  $ n4: int [1:20] 0 1 1 1 0 0 0 1 0 2 ...
```

```
sim1
```

```
## $n1
##  [1] 0 1 0 1 2 0 0 1 0 0
##
## $n2
##  [1] 3 1 1 1 0 2 0 0 0 3
##
## $n3
##  [1] 1 1 1 3 1 1 0 0 0 0 2 1 1 1 0 0 1 0 0 0
##
## $n4
##  [1] 0 1 1 1 0 0 0 1 0 2 0 1 2 0 1 0 0 2 2 1
```

We see a list containg one variable for each row of the parameter grid, where all the generated data points are stored. For example, the last row (stored under `sim1$n4`) specified n = 10 draws from the normal distribution with $\lambda = 1$. Since we set `nrep = 1` in order to save space, the draw only happened once.

The format list offers alot of flexibility, but is not very overseeable. At this point we can use the raw data to run summary statistics on, which we´ll do in the next passage.

```
# Application to Uniform distribution
param_list_runif <- list(c("n", 10, 30, 10)
                        ,c("min", 0, 0, 0)
                        ,c("max", 1, 1, 0))



grid_unif <- create_grid(param_list_runif, nrep=3)
tail(data_generation(simulation=runif, grid=grid_unif),1)
```

```
## $n9
##  [1] 0.48204261 0.25296493 0.21625479 0.67437639 0.04766363 0.70085309
##  [7] 0.35188864 0.40894400 0.82095132 0.91885735 0.28252833 0.96110479
## [13] 0.72839443 0.68637508 0.05284394 0.39522013 0.47784538 0.56025326
## [19] 0.69826159 0.91568354 0.61835123 0.42842151 0.54208037 0.05847849
## [25] 0.26085686 0.39715195 0.19774474 0.83192756 0.15288722 0.80341854
```

```
# Application to Poisson distribution

param_list_rpois <- list(c("n", 10, 30, 10)
                         , c("lambda", 0, 10, 1))

grid_pois <- create_grid(param_list_rpois, nrep=3)
tail(grid_pois,2) # nrow(grid_pois) = 99
```

```
## # A tibble: 2 x 3
##       n lambda   rep
##   <dbl>  <dbl> <int>
## 1    30     10     2
## 2    30     10     3
```

```
tail(data_generation(simulation=rpois, grid=grid_pois),1)
```

```
## $n99
##  [1] 11  8 11  6  5  8 11  9 13 12  9  8 14 16  8 20  9  8 10  5 13 14  4  8 10
## [26] 10 10  7  9  7
```

2.3 Summary function

Using the tools we showed before the user is able to generate the *raw* data from an underlying distribution of his choice. The next step is to introduce a way of applying a defined summary statistics onto that data, which we realised using the function called `summary_function()`.

This function basically just applys the user defined summary function (under the input `sum_fun`) onto the raw data using a `sapply()`-loop. The results gets stored in a nrow(grid) x 1 dimensional matrix, which will be combined with the parameter grid in the next step, in order to correctly allocate each result to the related set of parameters.

```
#summary function for one input
summary_function <- function(sum_fun, data_input){

  count <- length(data_input)
  summary_matrix <- matrix(nrow=count, ncol=1)

  for(i in 1:count){
    input <- list(data_input[[i]])
    output <- sapply(sum_fun, do.call, input)
    summary_matrix[i] <- output
  }
```

```
  #output <- as.data.frame(summary_matrix)
  #names(output) <- sum_fun
  colnames(summary_matrix) <- sum_fun
  return(summary_matrix)
}
```

The example to demonstrate this function uses rnorm()-function as underlying DGB, where we specified a parameter grid over three parameters (n, μ and the standard deviation).

`summary_function` Example:

```
param_list3 <- list(c("n", 10, 20, 10)
                    ,c("mu", 1, 2, 0.25)
                    ,c("sd", 0.5, 1, 0.1))

grid_test <- create_grid(param_list3, nrep=3)
test_data <- data_generation(simulation=rnorm, grid=grid_test)
summary_data <- summary_function(sum_fun=list("mean"), data_input=test_data)
head(summary_data)
```

```
##           mean
## [1,] 1.0165396
## [2,] 0.8156316
## [3,] 1.0361343
## [4,] 1.0129142
## [5,] 1.0714922
## [6,] 1.1017460
```

2.4 Summary array funcation

Even tough we specified a fairly small parameter grid in the example above, our simulation consisted retuned 180 summarised data points for the specified simulation. In the main_function() the results from the previous step get merged with the parameter grid into one data frame. This way of storing the data allows the user to apply further data wrangling processes, but is not suitable for printing the output in a tidy and clear way. A multt-dimensional array is better suited for this case.

The function `create_array_function()` takes all relevant data from the steps before (parameter grid and the results of the Monte Carlo simulation) and transforms it into an array with the correct dimensions.

```
create_array_function <- function(comb, parameters, nrep){
  storage <- list()
```

```r
name_vec <- c()

for(i in 1:length(parameters)){
  #this creates the sequences of parameters
  a <- as.numeric(parameters[[i]][[2]])
  b <- as.numeric(parameters[[i]][[3]])
  c <- as.numeric(parameters[[i]][[4]])
  output <- seq(from=a, to=b, by=c)
  storage[[i]] <-  output
  name_vec[i] <- parameters[[i]][[1]]
  #this just stores the names of the variables
}


matrix.numeration <-  paste("rep","=", 1:nrep, sep = "")

if(length(parameters)==1){
  comb_ordered <-  comb %>% arrange(comb[,2])
  seq1 <- c(unlist(storage[1]))

  row.names <- paste(name_vec[1],"=",seq1, sep = "")

  dimension_array <- c(length(seq1), nrep)
  dim_names_list <- list(row.names, matrix.numeration)
}

if(length(parameters)==2){
  comb_ordered <-  comb %>% arrange(comb[,2])  %>% arrange(comb[,3])
  seq1 <- c(unlist(storage[1]))
  seq2 <- c(unlist(storage[2]))

  row.names <- paste(name_vec[1],"=",seq1, sep = "")
  column.names <-  paste(name_vec[2],"=",seq2, sep = "")

  dimension_array <- c(length(seq1), length(seq2), nrep)
  dim_names_list <- list(row.names, column.names, matrix.numeration)
}

if(length(parameters)==3){
  comb_ordered <-  comb %>% arrange(comb[,2])  %>%
    arrange(comb[,3]) %>% arrange(comb[,4])
  seq1 <- c(unlist(storage[1]))
```

```
    seq2 <- c(unlist(storage[2]))
    seq3 <- c(unlist(storage[3]))

    row.names <- paste(name_vec[1],"=",seq1, sep = "")
    column.names <-  paste(name_vec[2],"=",seq2, sep = "")
    matrix.names1 <-  paste(name_vec[3],"=",seq3, sep = "")

    dimension_array <- c(length(seq1), length(seq2), length(seq3), nrep)
    dim_names_list <- list(row.names, column.names,
                           matrix.names1, matrix.numeration)

  }



  array1 <- array(comb_ordered[,ncol(comb)]
                 #change to automatically adjust dim
                 , dim = dimension_array
                 , dim_names_list)
  return(array1)
}
```

In order to test this function, we need to set up an altered version of the main_function(), that is introduced in the next passage.

`create_array_function` Example:

```
# PREP TEST `create_array_function`
main_function_array_test <-  function(parameters #list of parameters
                                      , nrep #number of repetitions
                                      , simulation #data genereation
                                      , sum_fun){ #summary statistics


  grid <- create_grid(parameters, nrep) #Step 1: create grid
  raw_data <- data_generation(simulation, grid) #Step 2: simlate data
  summary <- summary_function(sum_fun, data_input=raw_data) #Step 3: Summary statistics
  comb <- cbind(grid, summary) #Step 4: Combine resuluts with parameters
  array_1 <- create_array_function(comb, parameters, nrep) #Step 5: Create array

  return(comb)
}

param_list3x <- list(c("n", 10, 20, 10)
                     ,c("mu", 0, 5, 1)
```

```r
                     ,c("sd", 0, 3, 1))

comb1 <- main_function_array_test(parameters=param_list3x
                                  , nrep = 3
                                  , simulation = rnorm
                                  , sum_fun="mean")

comb1
```

```
##       n mu sd rep        mean
## 1    10  0  0   1  0.00000000
## 2    10  0  0   2  0.00000000
## 3    10  0  0   3  0.00000000
## 4    10  0  1   1  0.35922134
## 5    10  0  1   2 -0.24529993
## 6    10  0  1   3  0.30127036
## 7    10  0  2   1 -0.87682021
## 8    10  0  2   2 -0.05139362
## 9    10  0  2   3  0.21500163
## 10   10  0  3   1 -0.34155391
## 11   10  0  3   2  1.39171300
## 12   10  0  3   3 -0.47740568
## 13   10  1  0   1  1.00000000
## 14   10  1  0   2  1.00000000
## 15   10  1  0   3  1.00000000
## 16   10  1  1   1  1.85606677
## 17   10  1  1   2  1.08353642
## 18   10  1  1   3  1.75209300
## 19   10  1  2   1  1.20131019
## 20   10  1  2   2  0.24266328
## 21   10  1  2   3  0.36576695
## 22   10  1  3   1  1.28434344
## 23   10  1  3   2  1.31732267
## 24   10  1  3   3  0.10709650
## 25   10  2  0   1  2.00000000
## 26   10  2  0   2  2.00000000
## 27   10  2  0   3  2.00000000
## 28   10  2  1   1  1.59764541
## 29   10  2  1   2  2.28341208
## 30   10  2  1   3  1.45181336
## 31   10  2  2   1  1.90453684
## 32   10  2  2   2  1.72994105
```

```
## 33  10  2  2   3  2.49244703
## 34  10  2  3   1  2.10033022
## 35  10  2  3   2  1.81042591
## 36  10  2  3   3  1.04130981
## 37  10  3  0   1  3.00000000
## 38  10  3  0   2  3.00000000
## 39  10  3  0   3  3.00000000
## 40  10  3  1   1  2.63120812
## 41  10  3  1   2  3.59385520
## 42  10  3  1   3  2.81574571
## 43  10  3  2   1  2.89743500
## 44  10  3  2   2  1.96899804
## 45  10  3  2   3  3.93754772
## 46  10  3  3   1  2.69908069
## 47  10  3  3   2  3.50098794
## 48  10  3  3   3  2.13559944
## 49  10  4  0   1  4.00000000
## 50  10  4  0   2  4.00000000
## 51  10  4  0   3  4.00000000
## 52  10  4  1   1  3.82219824
## 53  10  4  1   2  4.12258565
## 54  10  4  1   3  3.86034215
## 55  10  4  2   1  3.50970468
## 56  10  4  2   2  4.16381785
## 57  10  4  2   3  4.33227194
## 58  10  4  3   1  3.83854364
## 59  10  4  3   2  4.48136540
## 60  10  4  3   3  4.20084829
## 61  10  5  0   1  5.00000000
## 62  10  5  0   2  5.00000000
## 63  10  5  0   3  5.00000000
## 64  10  5  1   1  5.58999234
## 65  10  5  1   2  4.40116722
## 66  10  5  1   3  4.93374121
## 67  10  5  2   1  4.54233805
## 68  10  5  2   2  4.39439681
## 69  10  5  2   3  5.81643057
## 70  10  5  3   1  3.95939352
## 71  10  5  3   2  7.21864779
## 72  10  5  3   3  4.42680728
## 73  20  0  0   1  0.00000000
## 74  20  0  0   2  0.00000000
```

```
## 75  20  0  0   3  0.00000000
## 76  20  0  1   1 -0.20244373
## 77  20  0  1   2  0.42552405
## 78  20  0  1   3 -0.10368493
## 79  20  0  2   1 -0.68882519
## 80  20  0  2   2 -0.47500354
## 81  20  0  2   3  0.46671026
## 82  20  0  3   1  0.78807652
## 83  20  0  3   2  0.53057194
## 84  20  0  3   3 -0.65906976
## 85  20  1  0   1  1.00000000
## 86  20  1  0   2  1.00000000
## 87  20  1  0   3  1.00000000
## 88  20  1  1   1  1.09537899
## 89  20  1  1   2  0.83552932
## 90  20  1  1   3  1.06689950
## 91  20  1  2   1  1.39594429
## 92  20  1  2   2  0.96871781
## 93  20  1  2   3  0.87428345
## 94  20  1  3   1  1.00437565
## 95  20  1  3   2  0.57362188
## 96  20  1  3   3  0.75663145
## 97  20  2  0   1  2.00000000
## 98  20  2  0   2  2.00000000
## 99  20  2  0   3  2.00000000
## 100 20  2  1   1  2.20798366
## 101 20  2  1   2  1.80481173
## 102 20  2  1   3  1.72778263
## 103 20  2  2   1  2.14818480
## 104 20  2  2   2  1.95185491
## 105 20  2  2   3  1.61629923
## 106 20  2  3   1  2.04821244
## 107 20  2  3   2  1.83153637
## 108 20  2  3   3  2.34136536
## 109 20  3  0   1  3.00000000
## 110 20  3  0   2  3.00000000
## 111 20  3  0   3  3.00000000
## 112 20  3  1   1  2.85353958
## 113 20  3  1   2  3.00748426
## 114 20  3  1   3  2.84416720
## 115 20  3  2   1  2.96662484
## 116 20  3  2   2  2.66609826
```

```
## 117 20   3   2    3  3.02846509
## 118 20   3   3    1  2.97808487
## 119 20   3   3    2  3.34593868
## 120 20   3   3    3  4.16159922
## 121 20   4   0    1  4.00000000
## 122 20   4   0    2  4.00000000
## 123 20   4   0    3  4.00000000
## 124 20   4   1    1  3.70112620
## 125 20   4   1    2  3.75204660
## 126 20   4   1    3  4.28435094
## 127 20   4   2    1  4.06323402
## 128 20   4   2    2  4.01638330
## 129 20   4   2    3  3.84577068
## 130 20   4   3    1  3.92332967
## 131 20   4   3    2  4.26109438
## 132 20   4   3    3  3.83306193
## 133 20   5   0    1  5.00000000
## 134 20   5   0    2  5.00000000
## 135 20   5   0    3  5.00000000
## 136 20   5   1    1  5.10540144
## 137 20   5   1    2  5.01593857
## 138 20   5   1    3  4.87631310
## 139 20   5   2    1  5.05969398
## 140 20   5   2    2  4.27376597
## 141 20   5   2    3  5.45661476
## 142 20   5   3    1  4.81424850
## 143 20   5   3    2  6.30435336
## 144 20   5   3    3  4.15455760
```

```
create_array_function(comb=comb1, parameters=param_list3x, nrep=3)
```

```
## , , sd=0, rep=1
##
##      mu=0 mu=1 mu=2 mu=3 mu=4 mu=5
## n=10    0    1    2    3    4    5
## n=20    0    1    2    3    4    5
##
## , , sd=1, rep=1
##
##            mu=0     mu=1     mu=2     mu=3     mu=4     mu=5
## n=10  0.3592213 1.856067 1.597645 2.631208 3.822198 5.589992
## n=20 -0.2024437 1.095379 2.207984 2.853540 3.701126 5.105401
##
```

```
## , , sd=2, rep=1
##
##             mu=0     mu=1     mu=2     mu=3     mu=4     mu=5
## n=10 -0.8768202 1.201310 1.904537 2.897435 3.509705 4.542338
## n=20 -0.6888252 1.395944 2.148185 2.966625 4.063234 5.059694
##
## , , sd=3, rep=1
##
##             mu=0     mu=1     mu=2     mu=3     mu=4     mu=5
## n=10 -0.3415539 1.284343 2.100330 2.699081 3.838544 3.959394
## n=20  0.7880765 1.004376 2.048212 2.978085 3.923330 4.814248
##
## , , sd=0, rep=2
##
##      mu=0 mu=1 mu=2 mu=3 mu=4 mu=5
## n=10    0    1    2    3    4    5
## n=20    0    1    2    3    4    5
##
## , , sd=1, rep=2
##
##             mu=0      mu=1     mu=2     mu=3     mu=4     mu=5
## n=10 -0.2452999 1.0835364 2.283412 3.593855 4.122586 4.401167
## n=20  0.4255240 0.8355293 1.804812 3.007484 3.752047 5.015939
##
## , , sd=2, rep=2
##
##              mu=0      mu=1     mu=2     mu=3     mu=4     mu=5
## n=10 -0.05139362 0.2426633 1.729941 1.968998 4.163818 4.394397
## n=20 -0.47500354 0.9687178 1.951855 2.666098 4.016383 4.273766
##
## , , sd=3, rep=2
##
##            mu=0      mu=1     mu=2     mu=3     mu=4     mu=5
## n=10 1.3917130 1.3173227 1.810426 3.500988 4.481365 7.218648
## n=20 0.5305719 0.5736219 1.831536 3.345939 4.261094 6.304353
##
## , , sd=0, rep=3
##
##      mu=0 mu=1 mu=2 mu=3 mu=4 mu=5
## n=10    0    1    2    3    4    5
## n=20    0    1    2    3    4    5
##
```

```
## , , sd=1, rep=3
##
##              mu=0     mu=1     mu=2     mu=3     mu=4     mu=5
## n=10  0.3012704 1.752093 1.451813 2.815746 3.860342 4.933741
## n=20 -0.1036849 1.066900 1.727783 2.844167 4.284351 4.876313
##
## , , sd=2, rep=3
##
##             mu=0      mu=1     mu=2     mu=3     mu=4     mu=5
## n=10 0.2150016 0.3657670 2.492447 3.937548 4.332272 5.816431
## n=20 0.4667103 0.8742835 1.616299 3.028465 3.845771 5.456615
##
## , , sd=3, rep=3
##
##              mu=0      mu=1     mu=2     mu=3     mu=4     mu=5
## n=10 -0.4774057 0.1070965 1.041310 2.135599 4.200848 4.426807
## n=20 -0.6590698 0.7566315 2.341365 4.161599 3.833062 4.154558
```

We see, that an array with the right dimensions is created.

2.5 Average function

```r
average_function <- function(grid_for_avg, summary, nrep){
  grid_for_avg <- grid_for_avg[-ncol(grid_for_avg)] #remove column for reps
  n_rows <- nrow(grid_for_avg)
  n_col <- ncol(grid_for_avg)

  for(i in 1:n_rows){
    start <- 1 + (i-1)*nrep
    end <- i*nrep
    grid_for_avg[i, n_col+1] <- mean(summary[start:end, ])
  }

  grid_plus_mc <- data.frame(grid_for_avg)

  colnames(grid_plus_mc)[n_col+1] <- "avg"

  return(grid_plus_mc)

}
```

2.6 Output Function

Goal is to create a function, that takes the Monte Carlo simulation results and all parameter input and converts it onto output format that prints nicely into the console. `Output_function` created for this purpose to store simulation results. `array_1`,`average_over_reps`,`parameters`,`cores`,`simulation`, `nrep`,`cpt` are used as input parameters in the `output_function`. Except the parameter `cpt`, other parameters are defined and explanied before this section. `cpt` parameters will be explanied in the section `Main Function`. It is basically saving the execution time of the simulation. Lets go to the each code line and explain them briefly.

Firstly, `out` object is created to store simulation results, averaged results and summary of the result in list `list()` format. The name `Eco` implemented as a class of the list `out`, since the spesific class had to implemented for the simulation results as a part of the task in visualisation. The results from `array_1` saved here as `out$results` also result for `average_over_reps` saved as `out$average`. Next, same class name is assigned to to `out$results` and `out$average` also default classes of the both list objects kept as a class. The reason is to prevent the future error while using the ggplot2 methods for simulation results. Because ggplot2 methos works only some spesific classses (data.frame ,etc.). After that the reporting part is created as a report of simulation result by using `cat()` function. At the end function returned to the object `out`.

```
output_function <- function(array_1,average_over_reps,parameters,cores,simulation,
                            nrep,cpt){

  out <- list()#Create a emptly list to store simulation result and average result.
  class(out) <- "Eco"  #We have to implement a class. I just gave a random name. "Eco"
  out$results <- array_1 # Saved the simulation result
  out$average <- average_over_reps # And this is the result from average function. All the r
  #Because output_function will return the list "out".

  #To us ggplot function Alex has created a average function that takes average of the simul
  class(out$average) <- c("Eco",class(out$average))#Again, name the class of the average res
  class(out$results) <- c("Eco",class(out$results))#Also same for the simulation result.

  if(cores>1){
    parallel = "Multisession"
  } else {
    parallel = "Sequential"
  }
  #This part is just a report. It will be shown at the end of the simulation result.
  text <-  cat("\n",
          "Repetition(nrep)      : ",nrep,"\n\n",
          "Parallelization Type  : ",parallel,"\n\n",
          "Number of Cores Used in  Parallelization : ",cores," out of",detectCores(),"\n\r
          "Input Parameters : ",paste(parameters),"\n\n",
```

```
            "Simulation Length :",length(array_1),"\n",
            "Minumum :",min(array_1),"\n",
            "Maximum :",max(array_1),"\n",
            "Mean    :", mean(array_1),"\n",
            "Median  :",median(array_1),"\n\n",
            "Execution Time of Monte Carlo Simulation",as.numeric(cpt),"secs \n\n",
            "Name of The Class :",class(out))

  return(out)
}
```

Output of `output_function` will be as same as the `main_function`.That's no further example is needed here. It will be covered in the next topic called "Main function".

2.7 Main Function

The "main_function" is a function that consist of the helper functions that created above. Here, all the helper functions are included and additionaly some commands and functions added to improve the simulation results. Here only additional arguments will be explained, since the helper functions are explained before.

First, `if()&else()` commands are added to check the number of cores are used in the main function is bigger than maximum number of the cores or not. Logically the computer cannot use the cores that doesn't exist. `max.cores` is a numeric object that stores the maximum number of the cores in the CPU. By using the function `detectCores()` from "parallel" packeage ,maximum number of the cores are stored in `max.cores`. The next is to check if the seed is provided by user or not. If the seed is not provided by the user , the function `sample.int()`generates a random number and uses it as a seed for reproducibility of the simulation. After the function `set.seed()` , `Sys.time()`function is implemented to check execution time of the simulation. `startTime` saves the startind time of the simulation ans endTime saves the ending time of the simulation. At the end , `startTime`is subtracted from `endtTime` and `cpt`is created to store execution time. As explained before ,cpt is used in `output_function` as a part of summary. Lastly, `plan()` function is used for the parallelisation to run the methods "sequential" or "multisession". "Sequential" runs the simulation with 1 core which means no parallelisation is used and "Multisession" runs the simulation in parallel by using the number of cores that provided by user. For more details please run the command `?future::plan` in RStudio.

```
main_function <-  function(parameters #list of parameters
                           , nrep #number of repetitions
                           , simulation #data genereation
                           , sum_fun #summary statistics
                           ,seed = NULL#Reproducibility
                           ,cores=NULL){
```

```r
#Number of cores
max.cores <- detectCores()
if(cores>max.cores){
  stop("Number of Cores cannot be bigger than total number of cores")
}
if(!is.null(seed)) {#Reproducibility
  set.seed(seed)}#If seed provided then set.seed takes the number
else {
  warning("No seed provided!", call. = FALSE)
  seed <- sample.int(10000, 1)#if its not provided then we generate random seed
  set.seed(seed)
  message("Random seed = ", seed, "\n")}


startTime <- Sys.time()#Starting time




grid <- create_grid(parameters, nrep) #Step 1: create grid

if(cores > 1){
  plan(multisession,workers = cores)
} else{
  plan(sequential)
}
suppressMessages(raw_data <- data_generation(simulation, grid))

summary <- summary_function(sum_fun, data_input=raw_data) #Step 3: Summary statistics

average_over_reps <- average_function(grid_for_avg=create_grid(parameters, 1), summary, nr

comb <- cbind(grid, summary) #Step 4: Combine resuluts with parameters

array_1 <- create_array_function(comb, parameters, nrep) #Step 5: Create array

endTime <- Sys.time()#Endtime

cpt <- endTime - startTime#Execution time

summary_1 <- output_function(array_1,average_over_reps,parameters,cores,simulation,
```

```
                            nrep,cpt)

return(summary_1)
}
```

Lets test the main function by using `rnorm` Monter Carlo simulation.

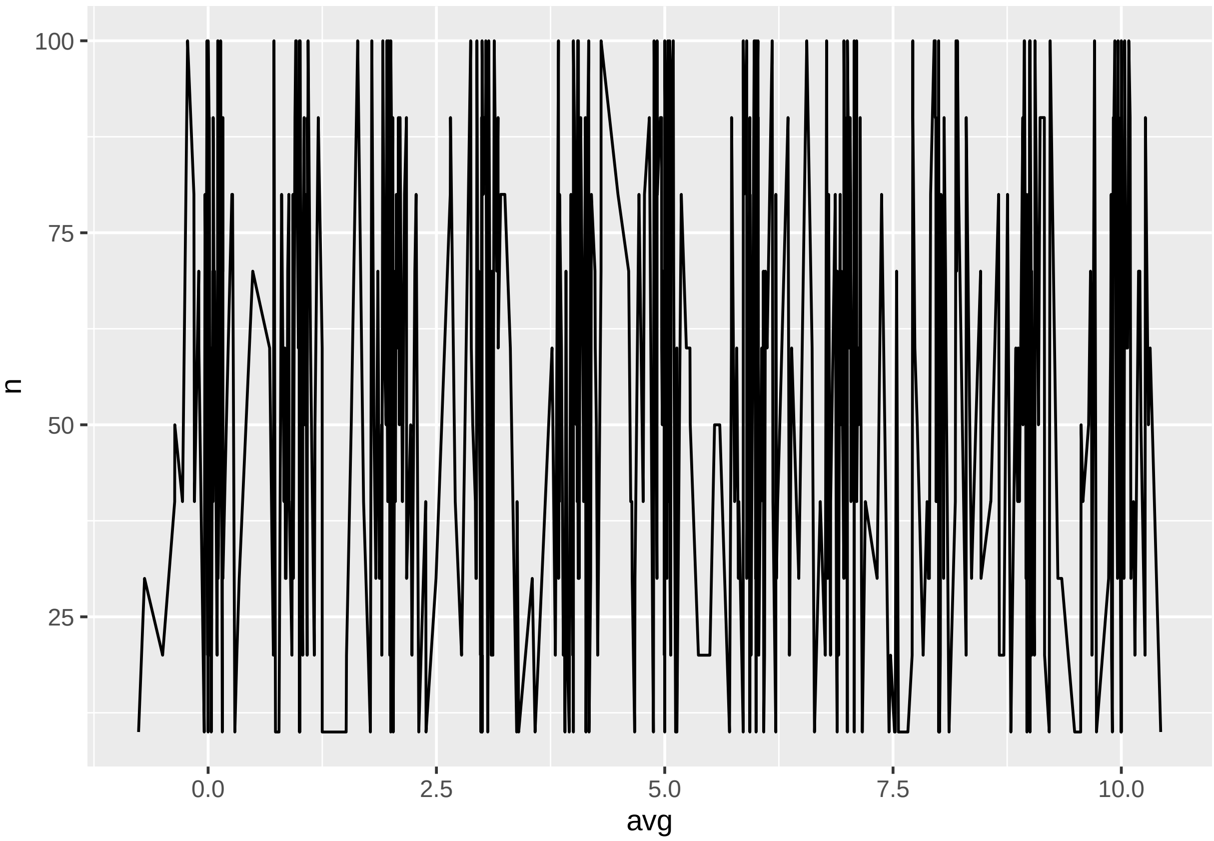
```
param_list3x <- list(c("n", 10, 100, 10)
                     ,c("mu", 0, 10, 1)
                     ,c("sd", 0, 5, 1))

test_me <- main_function(parameters=param_list3x
              , nrep = 5
              , simulation = rnorm
              , sum_fun="mean"
              ,seed=123
              ,cores=1)
```

```
##
##  Repetition(nrep)      :  5
##
##  Parallelization Type  :  Sequential
##
##  Number of Cores Used in  Parallelization :  1  out of 8
##
##  Input Parameters :  c("n", "10", "100", "10") c("mu", "0", "10", "1") c("sd", "0", "5",
##
##  Simulation Length : 3300
##  Minumum : -1.844634
##  Maximum : 13.43665
##  Mean     : 5.001471
##  Median   : 5
##
##  Execution Time of Monte Carlo Simulation 0.2983031 secs
##
##  Name of The Class : Eco
```

Here there are simulation results, average of simulation result and a summary about simulation. Now, lets check the `ggplot2` methods for this simulation results. `out$average` is created for visualisation purpose since working with arrays sometimes are trouble.`Ggplot2` methods can be used without any problem by taking average of the simulation.

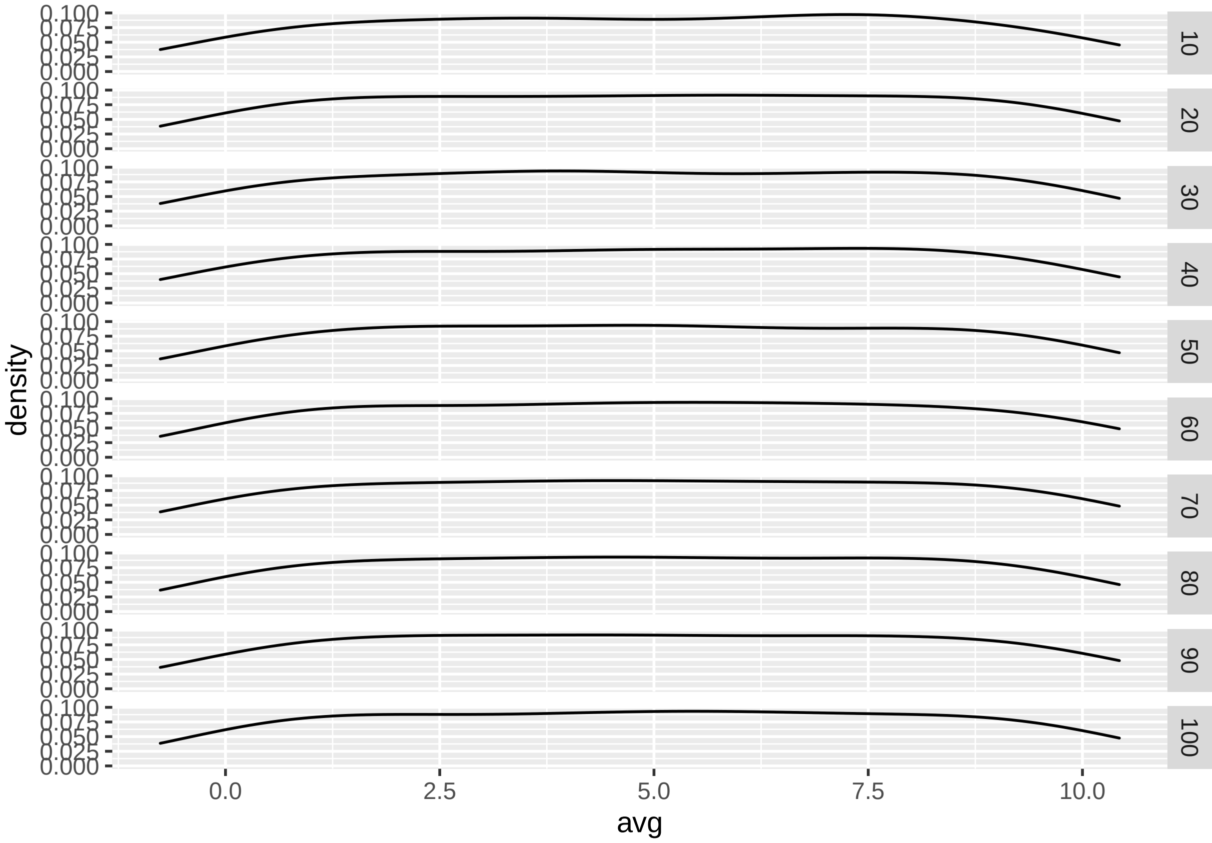
Lets try it.

```
ggplot(test_me$average,aes(x=avg,y=n))+geom_line()
```



Also with `facet_grid()` function.

```
ggplot(test_me$average,aes(x=avg))+facet_grid(n~.)+geom_density()
```



As proven above ,the simulation result works well with ggplot2 methods.

3 Examples

As an example to Monte Carlo Simulation , `OLS` and `GLS` *beta*2 coeffients are simulated by using parallelisation and also without parallelisation to show execution time of the parallel process.

```
ols_f <- function(n,mu,sd){
  e <- rnorm(n,mu,sd)
  x <- runif(n)
  y <- 0.5*x + e
  ols.hat <- t(x) %*% y / t(x)%*%x
  return("ols"=ols.hat)}

gls_f <- function(n,mu,sd){
  e <- rnorm(n,mu,sd)
  x <- runif(n)
  y <- 0.5*x + e
  v.inv <- diag(1/(1:n))
  c <- chol(v.inv)
  cy <- c %*% y
  cx <- c %*% x
  gls_hat <- t(cx) %*% cy / t(cx)%*%cx
```

```
  return("gls"=gls_hat)

  param_list <- list(c("n",100,1000,100),c("mu",0,1,0.25),c("sd",1,2,.5))
}
```

As seen as above, simple `OLS` and `GLS` functions are defined to find *beta*2 coefficients. Here, the execution time of the `GLS` function would be much more longer than `OLS` function.The reason for that in `GLS` function `The Cholesky Decomposition` is used.Lets run the simulation for `OLS` function.

Total execution time of `OLS` simulation is 8.46 seconds. Only one core is used.

```
 param_list <- list(c("n",100,1000,100),c("mu",0,1,0.25),c("sd",1,2,.5))
ols <- main_function(parameters = param_list,nrep=5,simulation = ols_f,sum_fun="mean",seed=1
```

```
##
##   Repetition(nrep)      :  5
##
##   Parallelization Type  :  Sequential
##
##   Number of Cores Used in  Parallelization :  1  out of 8
##
##   Input Parameters :  c("n", "100", "1000", "100") c("mu", "0", "1", "0.25") c("sd", "1",
##
##   Simulation Length : 750
##   Minumum : 0.08611764
##   Maximum : 2.598461
##   Mean    : 1.251827
##   Median  : 1.246948
##
##   Execution Time of Monte Carlo Simulation 0.09586287 secs
##
##   Name of The Class : Eco
```

```
ols
```

```
## $results
## , , sd=1, rep=1
##
##              mu=0   mu=0.25   mu=0.5  mu=0.75     mu=1
## n=100   0.6824267 0.7726517 1.390428 1.352546 2.033425
## n=200   0.4786798 0.7268500 1.551036 1.469126 2.016927
## n=300   0.5430408 0.6925565 1.412873 1.516396 2.020209
```

```
## n=400  0.4916201 0.9335676 1.240309 1.588818 1.939011
## n=500  0.4665197 0.8658038 1.302298 1.621465 2.071480
## n=600  0.4135505 1.0087766 1.263624 1.564343 1.873183
## n=700  0.6062303 0.8060657 1.423163 1.677571 1.915217
## n=800  0.5215150 0.8373901 1.384251 1.536015 1.845518
## n=900  0.4377088 0.8522425 1.323030 1.625229 2.113246
## n=1000 0.4272311 0.9598533 1.171133 1.607383 2.007037
##
## , , sd=1.5, rep=1
##
##             mu=0   mu=0.25   mu=0.5  mu=0.75     mu=1
## n=100  0.2325414 0.8709612 1.007916 1.187153 1.646406
## n=200  0.2684302 0.5956376 1.066756 1.572480 1.780621
## n=300  0.3463280 0.5074475 1.249930 1.526170 2.281022
## n=400  0.5532419 0.6727324 1.069168 1.587438 2.040513
## n=500  0.5088940 0.9917867 1.254739 1.687302 2.191762
## n=600  0.5817868 1.0516531 1.388582 1.787617 2.039833
## n=700  0.4085830 0.9266206 1.177307 1.618533 1.795094
## n=800  0.4788177 0.7297284 1.261184 1.635941 1.990439
## n=900  0.4293894 0.7521720 1.281461 1.550926 1.969204
## n=1000 0.4379182 0.7061176 1.281116 1.604609 1.875328
##
## , , sd=2, rep=1
##
##             mu=0   mu=0.25   mu=0.5  mu=0.75     mu=1
## n=100  0.6237415 0.8008305 1.639701 1.897970 2.513486
## n=200  0.6110261 1.2155626 1.815507 1.725145 2.033192
## n=300  0.4066040 0.8631502 1.239882 1.684178 2.106188
## n=400  0.4235146 0.6718906 1.072217 1.373997 2.000179
## n=500  0.4979046 1.0646659 1.248795 1.624880 2.036934
## n=600  0.4167380 0.9594940 1.222973 1.538267 1.972503
## n=700  0.4474413 0.7043636 1.318657 1.298269 2.086695
## n=800  0.3558261 0.9032952 1.340988 1.480559 1.852471
## n=900  0.4184591 0.8440885 1.312351 1.481778 1.959888
## n=1000 0.5482327 0.8486597 1.325121 1.557654 2.174067
##
## , , sd=1, rep=2
##
##             mu=0   mu=0.25   mu=0.5  mu=0.75     mu=1
## n=100  0.5699594 0.6347600 1.240574 1.409578 2.102247
## n=200  0.3411974 0.9859774 1.127496 1.654986 2.005527
## n=300  0.5568138 0.7710960 1.309760 1.640136 1.868297
```

```
## n=400  0.5892899 0.8630584 1.420656 1.488524 2.106760
## n=500  0.4565660 0.9875469 1.169863 1.642037 2.125870
## n=600  0.6074920 0.8311391 1.144387 1.561809 2.127146
## n=700  0.5311162 0.8793250 1.213227 1.758464 2.019087
## n=800  0.4907978 0.8137087 1.243065 1.544169 1.890569
## n=900  0.5006696 0.9126527 1.185916 1.551420 2.011440
## n=1000 0.5132049 0.8425093 1.323331 1.665086 2.067430
##
## , , sd=1.5, rep=2
##
##            mu=0   mu=0.25    mu=0.5  mu=0.75     mu=1
## n=100  0.4100172 1.0050342 1.3586338 1.594119 2.030257
## n=200  0.7621120 1.1129076 1.2050693 1.766159 2.052134
## n=300  0.4495498 0.9131922 1.3998834 1.844595 2.048539
## n=400  0.5189146 0.9392205 0.9085613 1.502305 2.159537
## n=500  0.7089984 0.6966299 1.1168839 1.472302 1.942223
## n=600  0.3991638 0.7265354 1.1572410 1.748560 1.735354
## n=700  0.4141086 0.7800295 1.3853238 1.765437 1.860513
## n=800  0.3797112 0.8637205 1.1967238 1.669677 1.904555
## n=900  0.4946693 0.9968749 1.3024325 1.747231 2.093768
## n=1000 0.6315810 0.8282354 1.1035843 1.541674 1.946093
##
## , , sd=2, rep=2
##
##            mu=0   mu=0.25    mu=0.5  mu=0.75     mu=1
## n=100  0.2670268 1.0417428 0.9765324 1.779177 2.139938
## n=200  0.2064741 0.8966665 1.2011619 1.562213 1.929578
## n=300  0.2953656 0.8280923 1.4380720 1.301385 2.089569
## n=400  0.7044943 0.8050854 1.4651276 1.320472 2.051137
## n=500  0.4125427 0.8432624 0.9844522 1.838044 2.144216
## n=600  0.7200719 0.9085732 1.0035755 1.809966 2.205790
## n=700  0.5352158 0.8738860 1.2227186 1.564884 2.114450
## n=800  0.3849922 0.7982686 1.2963208 1.625716 1.980291
## n=900  0.3505717 0.7606954 1.0653161 1.494651 1.986403
## n=1000 0.5160520 0.8734081 1.3842792 1.766722 2.085573
##
## , , sd=1, rep=3
##
##            mu=0   mu=0.25   mu=0.5  mu=0.75     mu=1
## n=100  0.5968253 1.0425781 1.207359 1.494982 1.755818
## n=200  0.6013728 0.8169956 1.089477 1.759701 2.050444
## n=300  0.4185383 0.8722548 1.202220 1.632546 2.054984
```

```
## n=400  0.5761413 0.9439121 1.215469 1.611440 2.019500
## n=500  0.6625234 0.8265606 1.301594 1.551010 1.973081
## n=600  0.6338517 0.9970815 1.257844 1.716586 2.047527
## n=700  0.4954136 0.8538731 1.202218 1.656197 1.921142
## n=800  0.4847987 0.8606272 1.280399 1.681161 1.939804
## n=900  0.4581839 0.8963640 1.363804 1.640401 2.005699
## n=1000 0.4964781 0.9341992 1.243164 1.696117 1.985360
##
## , , sd=1.5, rep=3
##
##             mu=0   mu=0.25   mu=0.5  mu=0.75     mu=1
## n=100  0.5659700 0.7903221 1.254210 1.886166 1.790293
## n=200  0.5821258 0.6511703 1.343681 1.782181 1.894019
## n=300  0.6550444 0.9367739 1.380756 1.705998 2.028275
## n=400  0.7048194 0.9633662 1.120887 1.688024 1.811344
## n=500  0.3923555 0.7626086 1.104116 1.537444 2.047353
## n=600  0.4717144 0.8688271 1.175261 1.826726 1.983990
## n=700  0.3871651 0.6201482 1.305263 1.669981 2.042779
## n=800  0.4834246 0.9357010 1.305622 1.523736 2.081715
## n=900  0.4433156 0.7916531 1.359010 1.580634 1.940405
## n=1000 0.4946876 0.8067392 1.480151 1.486763 1.900808
##
## , , sd=2, rep=3
##
##             mu=0   mu=0.25    mu=0.5  mu=0.75     mu=1
## n=100  0.4237920 0.5144245 0.4718755 2.004104 2.186295
## n=200  0.5777614 0.7299343 1.6298909 1.499530 1.929751
## n=300  0.6276662 0.8600749 0.9626478 2.011661 2.175176
## n=400  0.5785860 0.9158230 1.1544399 1.389809 2.169336
## n=500  0.2516702 0.9907678 1.0602298 1.532465 1.682624
## n=600  0.5165529 0.8616768 1.5189127 1.652623 1.941119
## n=700  0.4674128 0.7479103 1.2133761 1.738886 1.996975
## n=800  0.5090721 0.8062749 1.3959268 1.542798 1.994769
## n=900  0.4197560 0.8280591 1.2129929 1.504396 1.947569
## n=1000 0.6527630 0.8873246 1.2662086 1.656169 2.115128
##
## , , sd=1, rep=4
##
##             mu=0   mu=0.25    mu=0.5  mu=0.75     mu=1
## n=100  0.7359835 1.0087000 1.1157127 1.851686 2.286974
## n=200  0.5466285 0.8691083 0.9809655 1.677466 1.914859
## n=300  0.4092951 0.8421824 1.1834134 1.463671 2.105769
```

```
## n=400  0.4870693 0.8787365 1.3442915 1.641625 2.038680
## n=500  0.6247941 0.9737552 1.1814126 1.583607 1.981457
## n=600  0.5543342 0.9385104 1.2433161 1.676785 1.916412
## n=700  0.4178732 0.8808745 1.1952781 1.530920 1.960542
## n=800  0.5618498 0.8943310 1.2475019 1.686068 1.946561
## n=900  0.5052058 0.8785736 1.2679218 1.571624 2.009535
## n=1000 0.4550461 0.8622264 1.2189931 1.640547 2.027112
##
## , , sd=1.5, rep=4
##
##             mu=0   mu=0.25    mu=0.5  mu=0.75     mu=1
## n=100  0.4449720 0.8435500 1.5231357 1.528876 2.461235
## n=200  0.3352203 0.8038153 0.9593336 1.823112 2.149320
## n=300  0.4930487 0.6791231 1.0272955 1.574880 2.304307
## n=400  0.3806721 1.1714969 1.4056411 1.543264 2.258722
## n=500  0.4589277 0.7928819 1.5044037 1.632672 1.829121
## n=600  0.4884702 0.6958911 1.2883971 1.606977 2.109358
## n=700  0.5038252 0.8396545 1.4939591 1.598702 2.198315
## n=800  0.5515391 0.8256058 1.2838778 1.644795 2.020077
## n=900  0.4102779 0.7769137 1.2511958 1.567744 1.925168
## n=1000 0.6885496 0.9734467 1.2688095 1.525439 1.946295
##
## , , sd=2, rep=4
##
##             mu=0   mu=0.25    mu=0.5  mu=0.75     mu=1
## n=100  0.5717011 0.8796835 2.1004676 1.351107 2.598461
## n=200  1.1810891 0.4725781 0.9070217 1.490537 2.444452
## n=300  0.5900949 1.0569764 1.3134750 1.766744 2.089249
## n=400  0.7380901 0.4503391 1.0841655 1.704982 1.998472
## n=500  0.6158445 1.0384364 1.6255235 1.911106 1.868147
## n=600  0.6300586 1.1140343 1.1895053 1.822727 1.933571
## n=700  0.3625215 0.9113498 1.2522415 1.654996 1.981563
## n=800  0.4427674 0.8570775 1.1367413 1.796147 2.103874
## n=900  0.3816606 0.8423425 1.3168243 1.712630 2.102760
## n=1000 0.4265370 0.9495205 1.3021528 1.603875 1.875588
##
## , , sd=1, rep=5
##
##             mu=0   mu=0.25   mu=0.5  mu=0.75     mu=1
## n=100  0.2288956 0.9259850 1.345124 1.767008 1.904891
## n=200  0.4782098 0.9611524 1.326272 1.689215 2.050280
## n=300  0.4520819 0.9920557 1.238380 1.753581 2.087970
```

```
## n=400  0.4573970 1.1576259 1.246390 1.497974 2.019858
## n=500  0.5434649 0.9485599 1.191573 1.631702 2.049623
## n=600  0.4632066 0.8962950 1.266618 1.657375 1.924914
## n=700  0.3907963 0.8164093 1.155234 1.615976 1.949318
## n=800  0.4930138 0.9572131 1.295102 1.684709 1.936302
## n=900  0.4979275 0.8736681 1.214887 1.584737 1.985072
## n=1000 0.5301261 0.8977935 1.313722 1.630559 1.935747
##
## , , sd=1.5, rep=5
##
##            mu=0   mu=0.25   mu=0.5  mu=0.75     mu=1
## n=100  0.4688968 0.9435859 1.380401 1.586464 2.192555
## n=200  0.7831954 0.8507725 1.104056 1.524999 1.987862
## n=300  0.4957244 0.9953593 1.100958 1.596888 2.032147
## n=400  0.6377009 1.1108968 1.039524 1.621890 1.989716
## n=500  0.2757090 0.7425858 1.221020 1.687816 1.782811
## n=600  0.4674703 0.9888671 1.462157 1.805296 2.070940
## n=700  0.3602414 0.8718152 1.369915 1.868538 1.981321
## n=800  0.3886475 1.1178513 1.246394 1.612441 2.065348
## n=900  0.4814983 0.8016443 1.148795 1.443979 2.099358
## n=1000 0.4683020 0.8041379 1.013566 1.662726 1.873865
##
## , , sd=2, rep=5
##
##             mu=0   mu=0.25    mu=0.5  mu=0.75     mu=1
## n=100  0.08611764 0.9398614 0.9146867 1.669708 2.319204
## n=200  0.67138321 0.6832920 1.0754579 1.433456 2.068870
## n=300  0.44312742 1.2584762 1.3301563 1.650963 2.100398
## n=400  0.66455181 0.8364694 1.2167647 1.894041 1.989744
## n=500  0.41708387 0.8419375 1.1330271 1.713047 2.102631
## n=600  0.60004510 0.7267214 1.2282300 1.533603 2.044685
## n=700  0.42035536 0.8611750 1.3460957 1.744825 2.228779
## n=800  0.59792089 1.0213057 0.9896550 1.553520 1.600474
## n=900  0.51242473 0.7970967 1.1751429 1.646252 2.039864
## n=1000 0.56705287 0.8053208 1.2585926 1.673913 1.889833
##
## attr(,"class")
## [1] "Eco"   "array"
##
## $average
##        n   mu  sd       avg
## 1    100 0.00 1.0 0.5628181
```

```
## 2     100 0.00 1.5 0.4244795
## 3     100 0.00 2.0 0.3944758
## 4     100 0.25 1.0 0.8769350
## 5     100 0.25 1.5 0.8906907
## 6     100 0.25 2.0 0.8353085
## 7     100 0.50 1.0 1.2598397
## 8     100 0.50 1.5 1.3048593
## 9     100 0.50 2.0 1.2206527
## 10    100 0.75 1.0 1.5751601
## 11    100 0.75 1.5 1.5565558
## 12    100 0.75 2.0 1.7404132
## 13    100 1.00 1.0 2.0166709
## 14    100 1.00 1.5 2.0241493
## 15    100 1.00 2.0 2.3514768
## 16    200 0.00 1.0 0.4892177
## 17    200 0.00 1.5 0.5462168
## 18    200 0.00 2.0 0.6495468
## 19    200 0.25 1.0 0.8720167
## 20    200 0.25 1.5 0.8028606
## 21    200 0.25 2.0 0.7996067
## 22    200 0.50 1.0 1.2150492
## 23    200 0.50 1.5 1.1357791
## 24    200 0.50 2.0 1.3258080
## 25    200 0.75 1.0 1.6500989
## 26    200 0.75 1.5 1.6937863
## 27    200 0.75 2.0 1.5421762
## 28    200 1.00 1.0 2.0076074
## 29    200 1.00 1.5 1.9727911
## 30    200 1.00 2.0 2.0811684
## 31    300 0.00 1.0 0.4759540
## 32    300 0.00 1.5 0.4879391
## 33    300 0.00 2.0 0.4725716
## 34    300 0.25 1.0 0.8340291
## 35    300 0.25 1.5 0.8063792
## 36    300 0.25 2.0 0.9733540
## 37    300 0.50 1.0 1.2693292
## 38    300 0.50 1.5 1.2317647
## 39    300 0.50 2.0 1.2568465
## 40    300 0.75 1.0 1.6012659
## 41    300 0.75 1.5 1.6497061
## 42    300 0.75 2.0 1.6829863
## 43    300 1.00 1.0 2.0274458
```

```
## 44   300 1.00 1.5 2.1388580
## 45   300 1.00 2.0 2.1121160
## 46   400 0.00 1.0 0.5203035
## 47   400 0.00 1.5 0.5590698
## 48   400 0.00 2.0 0.6218474
## 49   400 0.25 1.0 0.9553801
## 50   400 0.25 1.5 0.9715426
## 51   400 0.25 2.0 0.7359215
## 52   400 0.50 1.0 1.2934233
## 53   400 0.50 1.5 1.1087563
## 54   400 0.50 2.0 1.1985429
## 55   400 0.75 1.0 1.5656764
## 56   400 0.75 1.5 1.5885842
## 57   400 0.75 2.0 1.5366603
## 58   400 1.00 1.0 2.0247618
## 59   400 1.00 1.5 2.0519664
## 60   400 1.00 2.0 2.0417733
## 61   500 0.00 1.0 0.5507736
## 62   500 0.00 1.5 0.4689769
## 63   500 0.00 2.0 0.4390092
## 64   500 0.25 1.0 0.9204453
## 65   500 0.25 1.5 0.7972986
## 66   500 0.25 2.0 0.9558140
## 67   500 0.50 1.0 1.2293479
## 68   500 0.50 1.5 1.2402327
## 69   500 0.50 2.0 1.2104054
## 70   500 0.75 1.0 1.6059639
## 71   500 0.75 1.5 1.6035074
## 72   500 0.75 2.0 1.7239084
## 73   500 1.00 1.0 2.0403021
## 74   500 1.00 1.5 1.9586539
## 75   500 1.00 2.0 1.9669104
## 76   600 0.00 1.0 0.5344870
## 77   600 0.00 1.5 0.4817211
## 78   600 0.00 2.0 0.5766933
## 79   600 0.25 1.0 0.9343605
## 80   600 0.25 1.5 0.8663548
## 81   600 0.25 2.0 0.9141000
## 82   600 0.50 1.0 1.2351578
## 83   600 0.50 1.5 1.2943275
## 84   600 0.50 2.0 1.2326393
## 85   600 0.75 1.0 1.6353797
```

```
## 86   600 0.75 1.5 1.7550350
## 87   600 0.75 2.0 1.6714372
## 88   600 1.00 1.0 1.9778364
## 89   600 1.00 1.5 1.9878951
## 90   600 1.00 2.0 2.0195340
## 91   700 0.00 1.0 0.4882859
## 92   700 0.00 1.5 0.4147847
## 93   700 0.00 2.0 0.4465893
## 94   700 0.25 1.0 0.8473095
## 95   700 0.25 1.5 0.8076536
## 96   700 0.25 2.0 0.8197369
## 97   700 0.50 1.0 1.2378238
## 98   700 0.50 1.5 1.3463535
## 99   700 0.50 2.0 1.2706179
## 100  700 0.75 1.0 1.6478257
## 101  700 0.75 1.5 1.7042384
## 102  700 0.75 2.0 1.6003720
## 103  700 1.00 1.0 1.9530612
## 104  700 1.00 1.5 1.9756043
## 105  700 1.00 2.0 2.0816926
## 106  800 0.00 1.0 0.5103950
## 107  800 0.00 1.5 0.4564280
## 108  800 0.00 2.0 0.4581157
## 109  800 0.25 1.0 0.8726540
## 110  800 0.25 1.5 0.8945214
## 111  800 0.25 2.0 0.8772444
## 112  800 0.50 1.0 1.2900639
## 113  800 0.50 1.5 1.2587603
## 114  800 0.50 2.0 1.2319263
## 115  800 0.75 1.0 1.6264243
## 116  800 0.75 1.5 1.6173179
## 117  800 0.75 2.0 1.5997479
## 118  800 1.00 1.0 1.9117509
## 119  800 1.00 1.5 2.0124269
## 120  800 1.00 2.0 1.9063757
## 121  900 0.00 1.0 0.4799391
## 122  900 0.00 1.5 0.4518301
## 123  900 0.00 2.0 0.4165744
## 124  900 0.25 1.0 0.8827002
## 125  900 0.25 1.5 0.8238516
## 126  900 0.25 2.0 0.8144564
## 127  900 0.50 1.0 1.2711118
```

```
## 128  900 0.50 1.5 1.2685788
## 129  900 0.50 2.0 1.2165253
## 130  900 0.75 1.0 1.5946823
## 131  900 0.75 1.5 1.5781026
## 132  900 0.75 2.0 1.5679413
## 133  900 1.00 1.0 2.0249983
## 134  900 1.00 1.5 2.0055806
## 135  900 1.00 2.0 2.0072967
## 136 1000 0.00 1.0 0.4844173
## 137 1000 0.00 1.5 0.5442077
## 138 1000 0.00 2.0 0.5421275
## 139 1000 0.25 1.0 0.8993163
## 140 1000 0.25 1.5 0.8237354
## 141 1000 0.25 2.0 0.8728468
## 142 1000 0.50 1.0 1.2540685
## 143 1000 0.50 1.5 1.2294453
## 144 1000 0.50 2.0 1.3072707
## 145 1000 0.75 1.0 1.6479383
## 146 1000 0.75 1.5 1.5642421
## 147 1000 0.75 2.0 1.6516665
## 148 1000 1.00 1.0 2.0045373
## 149 1000 1.00 1.5 1.9084779
## 150 1000 1.00 2.0 2.0280377
##
## attr(,"class")
## [1] "Eco"
```

Same function ,`OLS` is now used in parallel with 8 cores. Execution time of the simulation is 3.85 seconds.

Now lets run `GLS` function in parallel and check the execution time. Execution time of the simulation is 16.84 seconds (with 8 cores).

```
gls <- main_function(parameters = param_list,nrep=5,simulation = gls_f,sum_fun="mean",seed=1
```

```
##
##  Repetition(nrep)      :  5
##
##  Parallelization Type  :  Multisession
##
##  Number of Cores Used in  Parallelization :  8  out of 8
##
##  Input Parameters :  c("n", "100", "1000", "100") c("mu", "0", "1", "0.25") c("sd", "1",
```

```
##
##  Simulation Length : 750
##  Minumum : -0.9971
##  Maximum : 3.383637
##  Mean    : 1.260196
##  Median  : 1.284057
##
##  Execution Time of Monte Carlo Simulation 30.88275 secs
##
##  Name of The Class : Eco
```

```
gls
```

```
## $results
## , , sd=1, rep=1
##
##             mu=0   mu=0.25    mu=0.5  mu=0.75     mu=1
## n=100  0.7554538 0.5016876 1.7356494 1.517285 2.639110
## n=200  0.3903311 0.6408042 1.5194431 1.417126 3.255660
## n=300  0.8317798 0.8173362 1.2688679 1.366244 2.007371
## n=400  0.0664591 0.7887018 1.1664611 1.262650 1.601207
## n=500  0.7349604 0.8108023 1.5108027 1.624144 2.316178
## n=600  0.5483240 1.3362425 0.7466396 1.539507 1.775243
## n=700  1.1797722 0.7454456 1.5538655 1.596843 1.851457
## n=800  0.7388026 0.4151828 1.2355987 2.121480 1.617091
## n=900  0.3375558 0.5516673 0.9502070 1.037452 2.600048
## n=1000 0.2812896 0.8840261 1.4690083 1.667241 2.124077
##
## , , sd=1.5, rep=1
##
##               mu=0    mu=0.25    mu=0.5   mu=0.75     mu=1
## n=100   1.07566007 0.97703407 0.3943886 1.0410972 1.357513
## n=200   0.52383817 1.45057710 1.6489542 1.5278402 1.404216
## n=300   0.71906564 0.40330063 2.2602221 0.8222903 2.070699
## n=400   1.04256675 1.22061498 0.7880425 1.1109698 1.966145
## n=500  -0.07188688 1.41714029 1.4742135 1.9709113 3.005006
## n=600   0.87050124 1.16986970 0.8726417 1.1228528 1.821454
## n=700  -0.25848227 0.49203602 1.0950518 1.6813917 1.675552
## n=800   0.39160576 0.03376035 1.3268501 1.6087353 2.523682
## n=900   0.01281269 0.82989922 1.1917932 2.1448010 2.249365
## n=1000  1.00643264 0.23351513 1.7264966 1.7730168 1.716113
##
## , , sd=2, rep=1
```

```
##
##              mu=0   mu=0.25   mu=0.5   mu=0.75     mu=1
## n=100   0.1989645 0.9356581 1.650652 1.4279687 2.842366
## n=200   0.6770072 0.8534528 1.886844 1.1371663 2.086750
## n=300   0.9021613 1.3441820 1.592364 1.5281411 1.918323
## n=400   0.3678666 0.6749776 1.494146 1.0124086 2.125654
## n=500  -0.3538140 1.4641727 1.702553 2.0690877 1.771409
## n=600   0.2521821 0.8727736 1.281437 1.6461148 1.787155
## n=700  -0.5283604 1.1573474 1.301522 1.6284337 2.040922
## n=800   0.7162405 1.7569821 1.089098 1.8510202 2.008889
## n=900  -0.1466358 1.2663258 0.669074 0.9226198 1.983874
## n=1000  0.4922426 0.5592436 1.209339 0.5571104 2.177159
##
## , , sd=1, rep=2
##
##               mu=0   mu=0.25    mu=0.5  mu=0.75     mu=1
## n=100   0.93794659 0.3042632 0.8558104 1.395916 2.488330
## n=200   0.46865388 0.8026164 0.6673771 1.754976 2.196959
## n=300   0.21497718 0.6857743 1.4560339 1.591513 1.441236
## n=400   0.09099797 0.1139196 0.9989419 1.530588 1.901867
## n=500   0.26214018 1.0189866 0.9483061 1.929226 2.028602
## n=600   0.36515870 1.1206859 1.3254734 2.154065 2.398600
## n=700   0.84564182 0.6471012 1.4065464 1.768395 2.419217
## n=800   0.85872788 0.7206308 1.3704309 1.246906 2.218637
## n=900  -0.14990817 1.2264243 1.6872835 1.774656 2.095731
## n=1000  0.96179814 0.6660319 1.2241939 1.846178 2.384093
##
## , , sd=1.5, rep=2
##
##             mu=0   mu=0.25    mu=0.5  mu=0.75     mu=1
## n=100  0.7971403 0.6846694 1.4747453 1.376802 1.289544
## n=200  0.8722172 1.5062628 0.8667520 1.455997 3.187951
## n=300  1.1649019 1.7666435 0.9995124 1.495770 1.687811
## n=400  0.8440923 0.5933976 0.8222845 1.806278 2.256190
## n=500  0.5786570 1.1103728 1.9378712 1.644442 2.421673
## n=600  0.7493392 1.1661280 1.2064266 1.645635 1.537470
## n=700  0.8992186 1.0192023 1.0847103 1.817849 1.520385
## n=800  0.5199020 0.9399592 1.7241364 1.260431 1.886227
## n=900  1.0116356 1.1141843 1.4591379 2.097425 2.420061
## n=1000 1.1299320 0.6024359 0.3630059 2.020767 1.720216
##
## , , sd=2, rep=2
```

```
##
##                mu=0    mu=0.25     mu=0.5    mu=0.75      mu=1
## n=100  -0.01919950  0.10389729 -0.7181519  2.1975727 2.688336
## n=200   1.02896044 -0.06349928  1.7339415  1.9835128 1.816331
## n=300  -0.18610242  1.60434976  1.6070085 -0.3271213 1.931503
## n=400   1.41709802  1.20750797  1.5523077  1.6424103 1.171583
## n=500   1.23639159  0.54645719  1.6774379  1.3658300 3.363879
## n=600   0.39037287  1.42422862  0.5759413  2.1932974 1.854756
## n=700  -0.34830352  0.85511492  1.2548947  2.4213875 2.016490
## n=800  -0.08188814  1.27109005  1.7893126  0.9299721 1.717897
## n=900  -0.07990839  0.66567655  1.8002084  1.7681873 1.740396
## n=1000  0.88195270  0.50717811  1.2185675  2.1915303 1.744440
##
## , , sd=1, rep=3
##
##              mu=0   mu=0.25    mu=0.5  mu=0.75      mu=1
## n=100  0.2500595 1.3250897 0.9547569 2.293587 1.819439
## n=200  0.5471549 0.9566464 1.3296827 1.775252 1.804901
## n=300  1.0456930 1.4996928 1.4615289 1.399473 2.154331
## n=400  0.6342614 0.5517242 1.4421203 1.787110 2.553724
## n=500  0.5785579 1.0079606 1.1506459 1.573662 1.915940
## n=600  0.3589415 0.7270598 1.5240060 1.239381 1.971919
## n=700  0.7345540 0.2262226 1.2611827 1.103715 2.039746
## n=800  0.5015531 1.9152089 1.6893004 1.561901 2.198732
## n=900  0.1721376 1.0750490 1.1722128 1.835734 1.946778
## n=1000 0.5581321 0.7179216 1.0656947 1.773012 2.400244
##
## , , sd=1.5, rep=3
##
##                mu=0   mu=0.25    mu=0.5  mu=0.75      mu=1
## n=100   0.75515002 1.0532149 1.3675520 2.458531 2.511075
## n=200  -0.04934648 0.5496542 2.0479018 1.675846 1.142718
## n=300   1.00181000 1.6369489 0.7753943 2.638377 1.829983
## n=400   0.38708014 1.2533585 1.4714135 1.435519 2.018202
## n=500   0.37250451 0.7996507 1.2444854 1.892440 1.427025
## n=600   0.26431978 1.3546424 1.2058852 1.796262 2.147378
## n=700   1.12511433 0.6271908 0.8411113 1.638469 1.970271
## n=800   0.52208688 0.6764603 1.6023554 1.526346 1.906584
## n=900   0.22266670 0.5468360 1.5856621 2.134440 1.515266
## n=1000  0.20373892 1.9340035 1.2128314 1.337933 1.380179
##
## , , sd=2, rep=3
```

```
##
##             mu=0    mu=0.25      mu=0.5  mu=0.75      mu=1
## n=100  -0.5134825 -0.9971000  0.02609116 2.483622 2.1737654
## n=200   0.8564846  0.5270190  2.19874069 1.115522 2.0483072
## n=300   0.1357223  1.1346463  0.67885853 1.989233 2.3651937
## n=400  -0.1132654  1.5174377 -0.39911038 2.430706 2.7068852
## n=500   0.2692164  0.9373865  0.45341757 1.488270 2.0855372
## n=600   0.2767981  1.9185888  1.10986631 1.779951 2.2288551
## n=700   0.4175472  0.7176046  0.78665976 1.500209 2.3208915
## n=800   0.2236755  0.7400305  1.63497839 2.006660 2.2390314
## n=900   1.3383189  1.7077645  1.90542412 2.962268 1.5069244
## n=1000  0.2337000  0.4510541  0.72055597 2.240282 0.9580619
##
## , , sd=1, rep=4
##
##            mu=0     mu=0.25    mu=0.5  mu=0.75     mu=1
## n=100  1.3469860 -0.05560687 1.0932454 1.903516 2.156825
## n=200  0.3208913  0.96409270 1.0863966 1.386434 1.835235
## n=300  0.6078219  0.39871289 1.3616965 2.100878 1.933667
## n=400  0.6771807  1.05598360 1.5240453 2.261056 2.531636
## n=500  0.4919199  1.10827112 1.5332686 1.517603 1.843585
## n=600  0.5637876  0.85089642 1.8731358 1.617511 2.200501
## n=700  0.3582705  0.99200307 0.8214109 1.560444 2.128348
## n=800  0.5240053  0.94117803 1.2620739 1.546574 1.873964
## n=900  0.6295721  0.83325772 0.9917639 1.333128 1.971597
## n=1000 0.6248638  1.00187859 1.3589893 1.665165 1.630167
##
## , , sd=1.5, rep=4
##
##             mu=0     mu=0.25    mu=0.5   mu=0.75     mu=1
## n=100  0.14190040  2.01821137 0.8517502 2.0724785 2.762258
## n=200  0.82223920  0.95236727 2.3576613 2.7525450 2.656997
## n=300  0.41468489  0.81135204 1.7321312 2.2129173 1.586342
## n=400  0.47456049  1.28481297 1.6045288 1.6357618 2.517795
## n=500  0.19596935 -0.09776629 1.3816823 1.1821549 1.806055
## n=600  0.04919382  1.33474572 1.2062848 0.0429013 2.320981
## n=700  0.34423613  0.48523985 1.9450566 1.5101308 2.451672
## n=800  1.32565822  0.84774132 1.2387826 2.3976442 1.870370
## n=900  0.47854733  0.51916320 1.1726107 1.6816128 1.705770
## n=1000 0.82817973  0.80225623 1.5175430 1.4264566 1.863020
##
## , , sd=2, rep=4
```

```
##
##             mu=0    mu=0.25     mu=0.5  mu=0.75     mu=1
## n=100  0.8525966  1.1749948 3.38363655 1.248533 3.057232
## n=200  0.5182802  0.4838462 1.41934260 2.241456 2.358571
## n=300  1.0733668  2.0578513 0.96974173 2.245360 1.290691
## n=400  0.8011078  0.4410973 0.90802926 1.296971 2.153393
## n=500  0.2470769  0.7742284 2.22385975 2.482988 1.599441
## n=600  1.1095269  0.4747139 0.06967184 1.576253 1.038778
## n=700  0.3443959 -0.7980158 0.88632336 1.763624 1.642792
## n=800  0.3212259 -0.1889027 1.12345703 2.337993 1.622583
## n=900  0.8488329  1.3243651 0.42830492 1.324446 2.120547
## n=1000 1.6411901  0.1763252 1.43712633 1.095596 1.415895
##
## , , sd=1, rep=5
##
##             mu=0   mu=0.25   mu=0.5  mu=0.75     mu=1
## n=100  0.1106945 1.3324074 1.283149 1.795427 1.479256
## n=200  0.5031188 1.5027343 1.460474 1.921770 2.617298
## n=300  0.3730896 0.6590489 1.300485 1.664153 2.229097
## n=400  0.4404047 1.6079649 1.600920 1.597121 1.219695
## n=500  0.9142947 0.8821124 0.993282 2.142944 2.017532
## n=600  0.7455402 1.3534103 1.252659 1.654961 2.019734
## n=700  0.1307611 0.5045976 1.915240 1.719891 1.816085
## n=800  0.0348579 0.4676373 1.687917 1.773396 1.871487
## n=900  0.5683141 1.0347100 1.329680 1.971053 2.031339
## n=1000 0.2944263 0.7265625 1.342830 1.397584 1.678541
##
## , , sd=1.5, rep=5
##
##              mu=0     mu=0.25    mu=0.5   mu=0.75     mu=1
## n=100   0.3492151 0.941764185 1.9927388 1.1271039 2.343358
## n=200   0.9322249 0.009236562 1.6216833 1.5246716 2.126926
## n=300  -0.2061382 0.527637441 1.1785690 1.2855802 1.955373
## n=400  -0.2080310 0.865475626 1.5810396 1.8863548 2.607916
## n=500   1.0591201 1.269133681 1.3106256 1.6893725 2.221825
## n=600   1.0529351 0.935277255 0.8883463 0.4731693 2.883712
## n=700  -0.1566764 1.060909722 1.3480097 1.5789494 2.593332
## n=800   0.3458820 0.824462226 1.0043539 1.3977273 1.529786
## n=900   0.4578455 0.443865375 1.0857179 1.1477187 1.641970
## n=1000  0.7074108 0.556170334 1.5201499 1.3028712 2.580936
##
## , , sd=2, rep=5
```

```
##
##                mu=0    mu=0.25    mu=0.5    mu=0.75      mu=1
## n=100    0.35838129  1.9134632 0.6236571  1.8050059 2.775072
## n=200    1.13555545 -0.3462561 0.5517002  1.2833010 2.082454
## n=300   -0.12779402  0.8027196 0.2661467  1.1407255 2.706796
## n=400    0.84449836  1.5107112 1.3472610  2.1182648 2.410341
## n=500    0.58335229  0.3130528 2.2292441  0.5206100 1.407526
## n=600    1.00687281  0.8306865 0.8167926 -0.4327573 1.897837
## n=700    0.90763719  0.7547756 1.2602192  1.9296762 2.212971
## n=800    1.16899109  0.5890770 1.1466515  0.6023830 1.086282
## n=900   -0.53156942  0.2064395 1.3326650  0.1082137 1.878707
## n=1000 -0.06566813  1.5126884 1.9075839  1.1067314 1.610853
##
## attr(,"class")
## [1] "Eco"   "array"
##
## $average
##        n   mu  sd       avg
## 1    100 0.00 1.0 0.6802281
## 2    100 0.00 1.5 0.6238132
## 3    100 0.00 2.0 0.1754521
## 4    100 0.25 1.0 0.6815682
## 5    100 0.25 1.5 1.1349788
## 6    100 0.25 2.0 0.6261827
## 7    100 0.50 1.0 1.1845221
## 8    100 0.50 1.5 1.2162350
## 9    100 0.50 2.0 0.9931770
## 10   100 0.75 1.0 1.7811463
## 11   100 0.75 1.5 1.6152025
## 12   100 0.75 2.0 1.8325404
## 13   100 1.00 1.0 2.1165920
## 14   100 1.00 1.5 2.0527497
## 15   100 1.00 2.0 2.7073544
## 16   200 0.00 1.0 0.4460300
## 17   200 0.00 1.5 0.6202346
## 18   200 0.00 2.0 0.8432576
## 19   200 0.25 1.0 0.9733788
## 20   200 0.25 1.5 0.8936196
## 21   200 0.25 2.0 0.2909125
## 22   200 0.50 1.0 1.2126747
## 23   200 0.50 1.5 1.7085905
## 24   200 0.50 2.0 1.5581137
```

```
## 25   200 0.75 1.0 1.6511116
## 26   200 0.75 1.5 1.7873799
## 27   200 0.75 2.0 1.5521916
## 28   200 1.00 1.0 2.3420107
## 29   200 1.00 1.5 2.1037616
## 30   200 1.00 2.0 2.0784825
## 31   300 0.00 1.0 0.6146723
## 32   300 0.00 1.5 0.6188648
## 33   300 0.00 2.0 0.3594708
## 34   300 0.25 1.0 0.8121130
## 35   300 0.25 1.5 1.0291765
## 36   300 0.25 2.0 1.3887498
## 37   300 0.50 1.0 1.3697225
## 38   300 0.50 1.5 1.3891658
## 39   300 0.50 2.0 1.0228240
## 40   300 0.75 1.0 1.6244521
## 41   300 0.75 1.5 1.6909870
## 42   300 0.75 2.0 1.3152675
## 43   300 1.00 1.0 1.9531402
## 44   300 1.00 1.5 1.8260415
## 45   300 1.00 2.0 2.0425014
## 46   400 0.00 1.0 0.3818608
## 47   400 0.00 1.5 0.5080537
## 48   400 0.00 2.0 0.6634611
## 49   400 0.25 1.0 0.8236588
## 50   400 0.25 1.5 1.0435319
## 51   400 0.25 2.0 1.0703464
## 52   400 0.50 1.0 1.3464978
## 53   400 0.50 1.5 1.2534618
## 54   400 0.50 2.0 0.9805268
## 55   400 0.75 1.0 1.6877048
## 56   400 0.75 1.5 1.5749766
## 57   400 0.75 2.0 1.7001523
## 58   400 1.00 1.0 1.9616256
## 59   400 1.00 1.5 2.2732495
## 60   400 1.00 2.0 2.1135711
## 61   500 0.00 1.0 0.5963746
## 62   500 0.00 1.5 0.4268728
## 63   500 0.00 2.0 0.3964446
## 64   500 0.25 1.0 0.9656266
## 65   500 0.25 1.5 0.8997062
## 66   500 0.25 2.0 0.8070595
```

```
## 67   500 0.50 1.0 1.2272611
## 68   500 0.50 1.5 1.4697756
## 69   500 0.50 2.0 1.6573025
## 70   500 0.75 1.0 1.7575159
## 71   500 0.75 1.5 1.6758640
## 72   500 0.75 2.0 1.5853571
## 73   500 1.00 1.0 2.0243674
## 74   500 1.00 1.5 2.1763168
## 75   500 1.00 2.0 2.0455584
## 76   600 0.00 1.0 0.5163504
## 77   600 0.00 1.5 0.5972578
## 78   600 0.00 2.0 0.6071506
## 79   600 0.25 1.0 1.0776590
## 80   600 0.25 1.5 1.1921326
## 81   600 0.25 2.0 1.1041983
## 82   600 0.50 1.0 1.3443827
## 83   600 0.50 1.5 1.0759169
## 84   600 0.50 2.0 0.7707418
## 85   600 0.75 1.0 1.6410851
## 86   600 0.75 1.5 1.0161640
## 87   600 0.75 2.0 1.3525717
## 88   600 1.00 1.0 2.0731992
## 89   600 1.00 1.5 2.1421991
## 90   600 1.00 2.0 1.7614761
## 91   700 0.00 1.0 0.6497999
## 92   700 0.00 1.5 0.3906821
## 93   700 0.00 2.0 0.1585833
## 94   700 0.25 1.0 0.6230740
## 95   700 0.25 1.5 0.7369157
## 96   700 0.25 2.0 0.5373653
## 97   700 0.50 1.0 1.3916491
## 98   700 0.50 1.5 1.2627879
## 99   700 0.50 2.0 1.0979239
## 100  700 0.75 1.0 1.5498575
## 101  700 0.75 1.5 1.6453580
## 102  700 0.75 2.0 1.8486661
## 103  700 1.00 1.0 2.0509706
## 104  700 1.00 1.5 2.0422424
## 105  700 1.00 2.0 2.0468134
## 106  800 0.00 1.0 0.5315894
## 107  800 0.00 1.5 0.6210270
## 108  800 0.00 2.0 0.4696490
```

```
## 109  800 0.25 1.0 0.8919676
## 110  800 0.25 1.5 0.6644767
## 111  800 0.25 2.0 0.8336554
## 112  800 0.50 1.0 1.4490643
## 113  800 0.50 1.5 1.3792957
## 114  800 0.50 2.0 1.3566995
## 115  800 0.75 1.0 1.6500513
## 116  800 0.75 1.5 1.6381768
## 117  800 0.75 2.0 1.5456057
## 118  800 1.00 1.0 1.9559821
## 119  800 1.00 1.5 1.9433300
## 120  800 1.00 2.0 1.7349365
## 121  900 0.00 1.0 0.3115343
## 122  900 0.00 1.5 0.4367016
## 123  900 0.00 2.0 0.2858076
## 124  900 0.25 1.0 0.9442217
## 125  900 0.25 1.5 0.6907896
## 126  900 0.25 2.0 1.0341143
## 127  900 0.50 1.0 1.2262295
## 128  900 0.50 1.5 1.2989843
## 129  900 0.50 2.0 1.2271353
## 130  900 0.75 1.0 1.5904047
## 131  900 0.75 1.5 1.8411995
## 132  900 0.75 2.0 1.4171470
## 133  900 1.00 1.0 2.1290987
## 134  900 1.00 1.5 1.9064864
## 135  900 1.00 2.0 1.8460896
## 136 1000 0.00 1.0 0.5441020
## 137 1000 0.00 1.5 0.7751388
## 138 1000 0.00 2.0 0.6366835
## 139 1000 0.25 1.0 0.7992841
## 140 1000 0.25 1.5 0.8256762
## 141 1000 0.25 2.0 0.6412979
## 142 1000 0.50 1.0 1.2921433
## 143 1000 0.50 1.5 1.2680054
## 144 1000 0.50 2.0 1.2986345
## 145 1000 0.75 1.0 1.6698362
## 146 1000 0.75 1.5 1.5722088
## 147 1000 0.75 2.0 1.4382501
## 148 1000 1.00 1.0 2.0434242
## 149 1000 1.00 1.5 1.8520927
## 150 1000 1.00 2.0 1.5812816
```

```
##
## attr(,"class")
## [1] "Eco"
```

Now without parallelization (with 1 core)

```
gls <- main_function(parameters = param_list,nrep=5,simulation = gls_f,sum_fun="mean",seed=1
```

```
##
##  Repetition(nrep)      :  5
##
##  Parallelization Type  :  Sequential
##
##  Number of Cores Used in  Parallelization :  1  out of 8
##
##  Input Parameters :  c("n", "100", "1000", "100") c("mu", "0", "1", "0.25") c("sd", "1",
##
##  Simulation Length : 750
##  Minumum : -0.9971
##  Maximum : 3.383637
##  Mean     : 1.260196
##  Median   : 1.284057
##
##  Execution Time of Monte Carlo Simulation 29.00723 secs
##
##  Name of The Class : Eco
```

```
gls
```

```
## $results
## , , sd=1, rep=1
##
##               mu=0   mu=0.25    mu=0.5  mu=0.75     mu=1
## n=100   0.7554538 0.5016876 1.7356494 1.517285 2.639110
## n=200   0.3903311 0.6408042 1.5194431 1.417126 3.255660
## n=300   0.8317798 0.8173362 1.2688679 1.366244 2.007371
## n=400   0.0664591 0.7887018 1.1664611 1.262650 1.601207
## n=500   0.7349604 0.8108023 1.5108027 1.624144 2.316178
## n=600   0.5483240 1.3362425 0.7466396 1.539507 1.775243
## n=700   1.1797722 0.7454456 1.5538655 1.596843 1.851457
## n=800   0.7388026 0.4151828 1.2355987 2.121480 1.617091
## n=900   0.3375558 0.5516673 0.9502070 1.037452 2.600048
```

```
## n=1000 0.2812896 0.8840261 1.4690083 1.667241 2.124077
##
## , , sd=1.5, rep=1
##
##               mu=0    mu=0.25    mu=0.5   mu=0.75     mu=1
## n=100   1.07566007 0.97703407 0.3943886 1.0410972 1.357513
## n=200   0.52383817 1.45057710 1.6489542 1.5278402 1.404216
## n=300   0.71906564 0.40330063 2.2602221 0.8222903 2.070699
## n=400   1.04256675 1.22061498 0.7880425 1.1109698 1.966145
## n=500  -0.07188688 1.41714029 1.4742135 1.9709113 3.005006
## n=600   0.87050124 1.16986970 0.8726417 1.1228528 1.821454
## n=700  -0.25848227 0.49203602 1.0950518 1.6813917 1.675552
## n=800   0.39160576 0.03376035 1.3268501 1.6087353 2.523682
## n=900   0.01281269 0.82989922 1.1917932 2.1448010 2.249365
## n=1000  1.00643264 0.23351513 1.7264966 1.7730168 1.716113
##
## , , sd=2, rep=1
##
##              mu=0   mu=0.25   mu=0.5   mu=0.75     mu=1
## n=100   0.1989645 0.9356581 1.650652 1.4279687 2.842366
## n=200   0.6770072 0.8534528 1.886844 1.1371663 2.086750
## n=300   0.9021613 1.3441820 1.592364 1.5281411 1.918323
## n=400   0.3678666 0.6749776 1.494146 1.0124086 2.125654
## n=500  -0.3538140 1.4641727 1.702553 2.0690877 1.771409
## n=600   0.2521821 0.8727736 1.281437 1.6461148 1.787155
## n=700  -0.5283604 1.1573474 1.301522 1.6284337 2.040922
## n=800   0.7162405 1.7569821 1.089098 1.8510202 2.008889
## n=900  -0.1466358 1.2663258 0.669074 0.9226198 1.983874
## n=1000  0.4922426 0.5592436 1.209339 0.5571104 2.177159
##
## , , sd=1, rep=2
##
##               mu=0   mu=0.25    mu=0.5  mu=0.75     mu=1
## n=100   0.93794659 0.3042632 0.8558104 1.395916 2.488330
## n=200   0.46865388 0.8026164 0.6673771 1.754976 2.196959
## n=300   0.21497718 0.6857743 1.4560339 1.591513 1.441236
## n=400   0.09099797 0.1139196 0.9989419 1.530588 1.901867
## n=500   0.26214018 1.0189866 0.9483061 1.929226 2.028602
## n=600   0.36515870 1.1206859 1.3254734 2.154065 2.398600
## n=700   0.84564182 0.6471012 1.4065464 1.768395 2.419217
## n=800   0.85872788 0.7206308 1.3704309 1.246906 2.218637
## n=900  -0.14990817 1.2264243 1.6872835 1.774656 2.095731
```

```
## n=1000  0.96179814 0.6660319 1.2241939 1.846178 2.384093
##
## , , sd=1.5, rep=2
##
##             mu=0   mu=0.25    mu=0.5  mu=0.75     mu=1
## n=100  0.7971403 0.6846694 1.4747453 1.376802 1.289544
## n=200  0.8722172 1.5062628 0.8667520 1.455997 3.187951
## n=300  1.1649019 1.7666435 0.9995124 1.495770 1.687811
## n=400  0.8440923 0.5933976 0.8222845 1.806278 2.256190
## n=500  0.5786570 1.1103728 1.9378712 1.644442 2.421673
## n=600  0.7493392 1.1661280 1.2064266 1.645635 1.537470
## n=700  0.8992186 1.0192023 1.0847103 1.817849 1.520385
## n=800  0.5199020 0.9399592 1.7241364 1.260431 1.886227
## n=900  1.0116356 1.1141843 1.4591379 2.097425 2.420061
## n=1000 1.1299320 0.6024359 0.3630059 2.020767 1.720216
##
## , , sd=2, rep=2
##
##               mu=0     mu=0.25     mu=0.5    mu=0.75     mu=1
## n=100  -0.01919950  0.10389729 -0.7181519  2.1975727 2.688336
## n=200   1.02896044 -0.06349928  1.7339415  1.9835128 1.816331
## n=300  -0.18610242  1.60434976  1.6070085 -0.3271213 1.931503
## n=400   1.41709802  1.20750797  1.5523077  1.6424103 1.171583
## n=500   1.23639159  0.54645719  1.6774379  1.3658300 3.363879
## n=600   0.39037287  1.42422862  0.5759413  2.1932974 1.854756
## n=700  -0.34830352  0.85511492  1.2548947  2.4213875 2.016490
## n=800  -0.08188814  1.27109005  1.7893126  0.9299721 1.717897
## n=900  -0.07990839  0.66567655  1.8002084  1.7681873 1.740396
## n=1000  0.88195270  0.50717811  1.2185675  2.1915303 1.744440
##
## , , sd=1, rep=3
##
##             mu=0   mu=0.25    mu=0.5  mu=0.75     mu=1
## n=100  0.2500595 1.3250897 0.9547569 2.293587 1.819439
## n=200  0.5471549 0.9566464 1.3296827 1.775252 1.804901
## n=300  1.0456930 1.4996928 1.4615289 1.399473 2.154331
## n=400  0.6342614 0.5517242 1.4421203 1.787110 2.553724
## n=500  0.5785579 1.0079606 1.1506459 1.573662 1.915940
## n=600  0.3589415 0.7270598 1.5240060 1.239381 1.971919
## n=700  0.7345540 0.2262226 1.2611827 1.103715 2.039746
## n=800  0.5015531 1.9152089 1.6893004 1.561901 2.198732
## n=900  0.1721376 1.0750490 1.1722128 1.835734 1.946778
```

```
## n=1000 0.5581321 0.7179216 1.0656947 1.773012 2.400244
##
## , , sd=1.5, rep=3
##
##                mu=0   mu=0.25    mu=0.5  mu=0.75     mu=1
## n=100   0.75515002 1.0532149 1.3675520 2.458531 2.511075
## n=200  -0.04934648 0.5496542 2.0479018 1.675846 1.142718
## n=300   1.00181000 1.6369489 0.7753943 2.638377 1.829983
## n=400   0.38708014 1.2533585 1.4714135 1.435519 2.018202
## n=500   0.37250451 0.7996507 1.2444854 1.892440 1.427025
## n=600   0.26431978 1.3546424 1.2058852 1.796262 2.147378
## n=700   1.12511433 0.6271908 0.8411113 1.638469 1.970271
## n=800   0.52208688 0.6764603 1.6023554 1.526346 1.906584
## n=900   0.22266670 0.5468360 1.5856621 2.134440 1.515266
## n=1000  0.20373892 1.9340035 1.2128314 1.337933 1.380179
##
## , , sd=2, rep=3
##
##               mu=0    mu=0.25      mu=0.5  mu=0.75      mu=1
## n=100  -0.5134825 -0.9971000  0.02609116 2.483622 2.1737654
## n=200   0.8564846  0.5270190  2.19874069 1.115522 2.0483072
## n=300   0.1357223  1.1346463  0.67885853 1.989233 2.3651937
## n=400  -0.1132654  1.5174377 -0.39911038 2.430706 2.7068852
## n=500   0.2692164  0.9373865  0.45341757 1.488270 2.0855372
## n=600   0.2767981  1.9185888  1.10986631 1.779951 2.2288551
## n=700   0.4175472  0.7176046  0.78665976 1.500209 2.3208915
## n=800   0.2236755  0.7400305  1.63497839 2.006660 2.2390314
## n=900   1.3383189  1.7077645  1.90542412 2.962268 1.5069244
## n=1000  0.2337000  0.4510541  0.72055597 2.240282 0.9580619
##
## , , sd=1, rep=4
##
##              mu=0     mu=0.25    mu=0.5  mu=0.75     mu=1
## n=100  1.3469860 -0.05560687 1.0932454 1.903516 2.156825
## n=200  0.3208913  0.96409270 1.0863966 1.386434 1.835235
## n=300  0.6078219  0.39871289 1.3616965 2.100878 1.933667
## n=400  0.6771807  1.05598360 1.5240453 2.261056 2.531636
## n=500  0.4919199  1.10827112 1.5332686 1.517603 1.843585
## n=600  0.5637876  0.85089642 1.8731358 1.617511 2.200501
## n=700  0.3582705  0.99200307 0.8214109 1.560444 2.128348
## n=800  0.5240053  0.94117803 1.2620739 1.546574 1.873964
## n=900  0.6295721  0.83325772 0.9917639 1.333128 1.971597
```

```
## n=1000 0.6248638  1.00187859 1.3589893 1.665165 1.630167
##
## , , sd=1.5, rep=4
##
##              mu=0     mu=0.25    mu=0.5   mu=0.75     mu=1
## n=100  0.14190040  2.01821137 0.8517502 2.0724785 2.762258
## n=200  0.82223920  0.95236727 2.3576613 2.7525450 2.656997
## n=300  0.41468489  0.81135204 1.7321312 2.2129173 1.586342
## n=400  0.47456049  1.28481297 1.6045288 1.6357618 2.517795
## n=500  0.19596935 -0.09776629 1.3816823 1.1821549 1.806055
## n=600  0.04919382  1.33474572 1.2062848 0.0429013 2.320981
## n=700  0.34423613  0.48523985 1.9450566 1.5101308 2.451672
## n=800  1.32565822  0.84774132 1.2387826 2.3976442 1.870370
## n=900  0.47854733  0.51916320 1.1726107 1.6816128 1.705770
## n=1000 0.82817973  0.80225623 1.5175430 1.4264566 1.863020
##
## , , sd=2, rep=4
##
##             mu=0    mu=0.25     mu=0.5  mu=0.75     mu=1
## n=100  0.8525966  1.1749948 3.38363655 1.248533 3.057232
## n=200  0.5182802  0.4838462 1.41934260 2.241456 2.358571
## n=300  1.0733668  2.0578513 0.96974173 2.245360 1.290691
## n=400  0.8011078  0.4410973 0.90802926 1.296971 2.153393
## n=500  0.2470769  0.7742284 2.22385975 2.482988 1.599441
## n=600  1.1095269  0.4747139 0.06967184 1.576253 1.038778
## n=700  0.3443959 -0.7980158 0.88632336 1.763624 1.642792
## n=800  0.3212259 -0.1889027 1.12345703 2.337993 1.622583
## n=900  0.8488329  1.3243651 0.42830492 1.324446 2.120547
## n=1000 1.6411901  0.1763252 1.43712633 1.095596 1.415895
##
## , , sd=1, rep=5
##
##             mu=0   mu=0.25   mu=0.5  mu=0.75     mu=1
## n=100  0.1106945 1.3324074 1.283149 1.795427 1.479256
## n=200  0.5031188 1.5027343 1.460474 1.921770 2.617298
## n=300  0.3730896 0.6590489 1.300485 1.664153 2.229097
## n=400  0.4404047 1.6079649 1.600920 1.597121 1.219695
## n=500  0.9142947 0.8821124 0.993282 2.142944 2.017532
## n=600  0.7455402 1.3534103 1.252659 1.654961 2.019734
## n=700  0.1307611 0.5045976 1.915240 1.719891 1.816085
## n=800  0.0348579 0.4676373 1.687917 1.773396 1.871487
## n=900  0.5683141 1.0347100 1.329680 1.971053 2.031339
```

```
## n=1000 0.2944263 0.7265625 1.342830 1.397584 1.678541
##
## , , sd=1.5, rep=5
##
##              mu=0     mu=0.25    mu=0.5   mu=0.75     mu=1
## n=100   0.3492151 0.941764185 1.9927388 1.1271039 2.343358
## n=200   0.9322249 0.009236562 1.6216833 1.5246716 2.126926
## n=300  -0.2061382 0.527637441 1.1785690 1.2855802 1.955373
## n=400  -0.2080310 0.865475626 1.5810396 1.8863548 2.607916
## n=500   1.0591201 1.269133681 1.3106256 1.6893725 2.221825
## n=600   1.0529351 0.935277255 0.8883463 0.4731693 2.883712
## n=700  -0.1566764 1.060909722 1.3480097 1.5789494 2.593332
## n=800   0.3458820 0.824462226 1.0043539 1.3977273 1.529786
## n=900   0.4578455 0.443865375 1.0857179 1.1477187 1.641970
## n=1000  0.7074108 0.556170334 1.5201499 1.3028712 2.580936
##
## , , sd=2, rep=5
##
##               mu=0    mu=0.25    mu=0.5    mu=0.75     mu=1
## n=100   0.35838129  1.9134632 0.6236571  1.8050059 2.775072
## n=200   1.13555545 -0.3462561 0.5517002  1.2833010 2.082454
## n=300  -0.12779402  0.8027196 0.2661467  1.1407255 2.706796
## n=400   0.84449836  1.5107112 1.3472610  2.1182648 2.410341
## n=500   0.58335229  0.3130528 2.2292441  0.5206100 1.407526
## n=600   1.00687281  0.8306865 0.8167926 -0.4327573 1.897837
## n=700   0.90763719  0.7547756 1.2602192  1.9296762 2.212971
## n=800   1.16899109  0.5890770 1.1466515  0.6023830 1.086282
## n=900  -0.53156942  0.2064395 1.3326650  0.1082137 1.878707
## n=1000 -0.06566813  1.5126884 1.9075839  1.1067314 1.610853
##
## attr(,"class")
## [1] "Eco"   "array"
##
## $average
##        n   mu  sd       avg
## 1    100 0.00 1.0 0.6802281
## 2    100 0.00 1.5 0.6238132
## 3    100 0.00 2.0 0.1754521
## 4    100 0.25 1.0 0.6815682
## 5    100 0.25 1.5 1.1349788
## 6    100 0.25 2.0 0.6261827
## 7    100 0.50 1.0 1.1845221
```

```
## 8     100 0.50 1.5 1.2162350
## 9     100 0.50 2.0 0.9931770
## 10    100 0.75 1.0 1.7811463
## 11    100 0.75 1.5 1.6152025
## 12    100 0.75 2.0 1.8325404
## 13    100 1.00 1.0 2.1165920
## 14    100 1.00 1.5 2.0527497
## 15    100 1.00 2.0 2.7073544
## 16    200 0.00 1.0 0.4460300
## 17    200 0.00 1.5 0.6202346
## 18    200 0.00 2.0 0.8432576
## 19    200 0.25 1.0 0.9733788
## 20    200 0.25 1.5 0.8936196
## 21    200 0.25 2.0 0.2909125
## 22    200 0.50 1.0 1.2126747
## 23    200 0.50 1.5 1.7085905
## 24    200 0.50 2.0 1.5581137
## 25    200 0.75 1.0 1.6511116
## 26    200 0.75 1.5 1.7873799
## 27    200 0.75 2.0 1.5521916
## 28    200 1.00 1.0 2.3420107
## 29    200 1.00 1.5 2.1037616
## 30    200 1.00 2.0 2.0784825
## 31    300 0.00 1.0 0.6146723
## 32    300 0.00 1.5 0.6188648
## 33    300 0.00 2.0 0.3594708
## 34    300 0.25 1.0 0.8121130
## 35    300 0.25 1.5 1.0291765
## 36    300 0.25 2.0 1.3887498
## 37    300 0.50 1.0 1.3697225
## 38    300 0.50 1.5 1.3891658
## 39    300 0.50 2.0 1.0228240
## 40    300 0.75 1.0 1.6244521
## 41    300 0.75 1.5 1.6909870
## 42    300 0.75 2.0 1.3152675
## 43    300 1.00 1.0 1.9531402
## 44    300 1.00 1.5 1.8260415
## 45    300 1.00 2.0 2.0425014
## 46    400 0.00 1.0 0.3818608
## 47    400 0.00 1.5 0.5080537
## 48    400 0.00 2.0 0.6634611
## 49    400 0.25 1.0 0.8236588
```

```
## 50   400 0.25 1.5 1.0435319
## 51   400 0.25 2.0 1.0703464
## 52   400 0.50 1.0 1.3464978
## 53   400 0.50 1.5 1.2534618
## 54   400 0.50 2.0 0.9805268
## 55   400 0.75 1.0 1.6877048
## 56   400 0.75 1.5 1.5749766
## 57   400 0.75 2.0 1.7001523
## 58   400 1.00 1.0 1.9616256
## 59   400 1.00 1.5 2.2732495
## 60   400 1.00 2.0 2.1135711
## 61   500 0.00 1.0 0.5963746
## 62   500 0.00 1.5 0.4268728
## 63   500 0.00 2.0 0.3964446
## 64   500 0.25 1.0 0.9656266
## 65   500 0.25 1.5 0.8997062
## 66   500 0.25 2.0 0.8070595
## 67   500 0.50 1.0 1.2272611
## 68   500 0.50 1.5 1.4697756
## 69   500 0.50 2.0 1.6573025
## 70   500 0.75 1.0 1.7575159
## 71   500 0.75 1.5 1.6758640
## 72   500 0.75 2.0 1.5853571
## 73   500 1.00 1.0 2.0243674
## 74   500 1.00 1.5 2.1763168
## 75   500 1.00 2.0 2.0455584
## 76   600 0.00 1.0 0.5163504
## 77   600 0.00 1.5 0.5972578
## 78   600 0.00 2.0 0.6071506
## 79   600 0.25 1.0 1.0776590
## 80   600 0.25 1.5 1.1921326
## 81   600 0.25 2.0 1.1041983
## 82   600 0.50 1.0 1.3443827
## 83   600 0.50 1.5 1.0759169
## 84   600 0.50 2.0 0.7707418
## 85   600 0.75 1.0 1.6410851
## 86   600 0.75 1.5 1.0161640
## 87   600 0.75 2.0 1.3525717
## 88   600 1.00 1.0 2.0731992
## 89   600 1.00 1.5 2.1421991
## 90   600 1.00 2.0 1.7614761
## 91   700 0.00 1.0 0.6497999
```

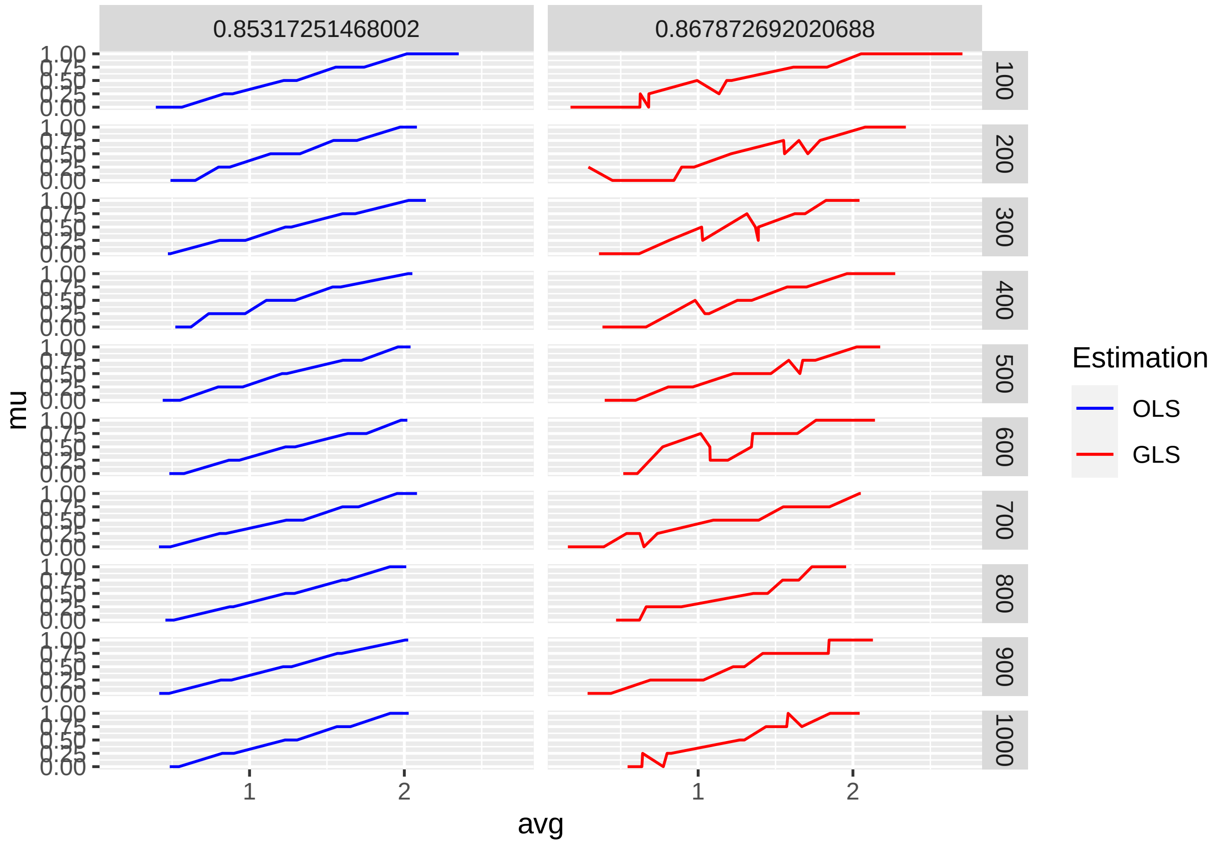
```
## 92   700 0.00 1.5 0.3906821
## 93   700 0.00 2.0 0.1585833
## 94   700 0.25 1.0 0.6230740
## 95   700 0.25 1.5 0.7369157
## 96   700 0.25 2.0 0.5373653
## 97   700 0.50 1.0 1.3916491
## 98   700 0.50 1.5 1.2627879
## 99   700 0.50 2.0 1.0979239
## 100  700 0.75 1.0 1.5498575
## 101  700 0.75 1.5 1.6453580
## 102  700 0.75 2.0 1.8486661
## 103  700 1.00 1.0 2.0509706
## 104  700 1.00 1.5 2.0422424
## 105  700 1.00 2.0 2.0468134
## 106  800 0.00 1.0 0.5315894
## 107  800 0.00 1.5 0.6210270
## 108  800 0.00 2.0 0.4696490
## 109  800 0.25 1.0 0.8919676
## 110  800 0.25 1.5 0.6644767
## 111  800 0.25 2.0 0.8336554
## 112  800 0.50 1.0 1.4490643
## 113  800 0.50 1.5 1.3792957
## 114  800 0.50 2.0 1.3566995
## 115  800 0.75 1.0 1.6500513
## 116  800 0.75 1.5 1.6381768
## 117  800 0.75 2.0 1.5456057
## 118  800 1.00 1.0 1.9559821
## 119  800 1.00 1.5 1.9433300
## 120  800 1.00 2.0 1.7349365
## 121  900 0.00 1.0 0.3115343
## 122  900 0.00 1.5 0.4367016
## 123  900 0.00 2.0 0.2858076
## 124  900 0.25 1.0 0.9442217
## 125  900 0.25 1.5 0.6907896
## 126  900 0.25 2.0 1.0341143
## 127  900 0.50 1.0 1.2262295
## 128  900 0.50 1.5 1.2989843
## 129  900 0.50 2.0 1.2271353
## 130  900 0.75 1.0 1.5904047
## 131  900 0.75 1.5 1.8411995
## 132  900 0.75 2.0 1.4171470
## 133  900 1.00 1.0 2.1290987
```

```
## 134  900 1.00 1.5 1.9064864
## 135  900 1.00 2.0 1.8460896
## 136 1000 0.00 1.0 0.5441020
## 137 1000 0.00 1.5 0.7751388
## 138 1000 0.00 2.0 0.6366835
## 139 1000 0.25 1.0 0.7992841
## 140 1000 0.25 1.5 0.8256762
## 141 1000 0.25 2.0 0.6412979
## 142 1000 0.50 1.0 1.2921433
## 143 1000 0.50 1.5 1.2680054
## 144 1000 0.50 2.0 1.2986345
## 145 1000 0.75 1.0 1.6698362
## 146 1000 0.75 1.5 1.5722088
## 147 1000 0.75 2.0 1.4382501
## 148 1000 1.00 1.0 2.0434242
## 149 1000 1.00 1.5 1.8520927
## 150 1000 1.00 2.0 1.5812816
##
## attr(,"class")
## [1] "Eco"
```

As seen as on the summary part, total execution time of the simulation took 36.35 seconds which also proves that parallel process works well.As a reminder, those simulations ran on MacBook Air 10 with total 8 (4 performance and 4 efficiency) cores. Execution times of the simulation might differ on other computers.

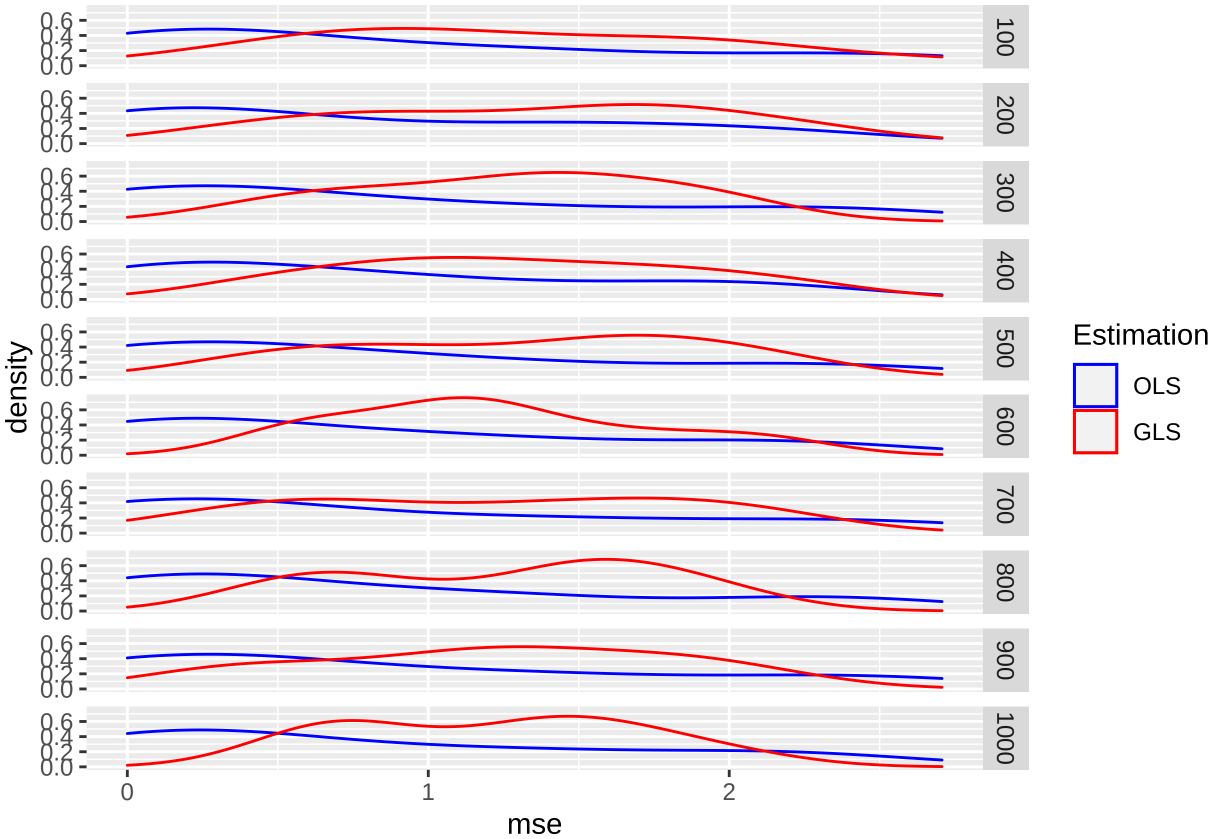
Now, lets use the simulation results and visualize them by using ggplot2 methods.Here, additional `MSE`(Mean Square Error) are calculated for each simulation and saved as `out$average$mse`.

```
gls$average <-  gls$average %>% mutate(mse =(2-avg)^2 )
ols$average <-  ols$average %>% mutate(mse =(2-avg)^2 )

ols$average %>% ggplot(aes(x=avg,y=mu,col="OLS"))+
  facet_grid(n~mean(mse))+geom_line()+
 geom_line(data=gls$average,aes(x=avg,y=mu,col="GLS"))+
  scale_color_manual(name = "Estimation", values = c("OLS" = "blue", "GLS" = "red"))
```



Also density graph of MSE for GLS and ‘OLS *beta*2 coefficients.

```
ggplot(ols$average,aes(x=mse,col="OLS"))+facet_grid(n~.)+geom_density()+
  geom_density(data=gls$average,aes(x=avg,col="GLS"))+
  scale_color_manual(name = "Estimation", values = c("OLS" = "blue", "GLS" = "red"))
```



4 Conclusion

The above section illustrates the power of our implemented model and gives the fairly easy to use tool, that still allows for a variety of different specifications in terms of used parameters, data generation processes and summary functions. Researchers, who use Monte Carlo studys on a regular basis, may save a lot of time using a tool like this in the long run.

By nature, there may be cases, where our implementation doesnt satisfy the needs of the user to the fullest, but for a wide variety of examples we showed, that it worked well and served the goal that we aimed for. Our functional programming approach allows for easy and flexible adjustments in case the use of our functions should be expanded, f.e. if a grid of more than 3 (or 4?) parameters is needed.

Theoretically, this work could be implemented as an R package to share it with the R community. But since the `MonteCarlo()` function of the `vigniette` package already provides a well working alternative to our project besides some minor differences, there is currently no need in doing that.

5 References

6 Contributions

Eidesstattliche Versicherung

Ich versichere an Eides statt durch meine Unterschrift, dass ich die vorstehende Arbeit selbständig und ohne fremde Hilfe angefertigt und alle Stellen, die ich wörtlich oder annähernd wörtlich aus Veröffentlichungen entnommen habe, als solche kenntlich gemacht habe, mich auch keiner anderen als der angegebenen Literatur oder sonstiger Hilfsmittel bedient habe. Die Arbeit hat in dieser oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen.

Essen, den ____________________ ______________________________

Alexander Langnau, Öcal Kaptan, Sunyoung Ji