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:	1st 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. The task in this paper is to create the wrapper functions providing a convenient interface for Monte Carlo Simulations.

2 Preprocess

2.1 Function for creating grid

`create_grid` is the function hat creates a parameter grid with all permutations of the given parameters. This is necessary to try all possible combinations to find the optimal parameters. This function tunes parameters to improve performance of Monte Carlo Simulation function.

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

  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() Example:

```
#One parameter (works)
param_list1 <- list(c("n", 10, 20, 10))
tail(create_grid(param_list1, nrep=10), 2)
```

```
## # A tibble: 2 x 2
##       n   rep
##   <dbl> <int>
## 1    20     9
## 2    20    10
```

```
tail(create_grid(param_list1, nrep=1), 2)
```

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

```
#two parameter (works)
param_list2 <- list(c("n", 10, 20, 10)
                    ,c("mu", 0, 1, 0.25))
tail(create_grid(param_list1, nrep=10), 2)
```

```
## # A tibble: 2 x 2
##       n   rep
##   <dbl> <int>
## 1    20     9
## 2    20    10
```

```
#three parameters (works)
param_list3 <- list(c("n", 10, 20, 10)
                    ,c("mu", 0, 1, 0.25)
                    ,c("sd", 0, 0.3, 0.1))
tail(create_grid(param_list3, nrep=10), 2)
```

```
## # A tibble: 2 x 4
```

```
##       n    mu    sd   rep
##   <dbl> <dbl> <dbl> <int>
## 1    20     1   0.3     9
## 2    20     1   0.3    10
```

```
#four parameters (works)
param_list4 <- list(c("n", 10, 20, 10)
                    ,c("mu", 0, 1, 0.25)
                    ,c("sd", 0, 0.3, 0.1)
                    ,c("gender", 0, 1, 1))

tail(create_grid(param_list4, nrep=5),2)
```

```
## # A tibble: 2 x 5
##       n    mu    sd gender   rep
##   <dbl> <dbl> <dbl>  <dbl> <int>
## 1    20     1   0.3      1     4
## 2    20     1   0.3      1     5
```

```
grid_4 <- create_grid(param_list4, nrep=50)
tail(grid_4,2)
```

```
## # A tibble: 2 x 5
##       n    mu    sd gender   rep
##   <dbl> <dbl> <dbl>  <dbl> <int>
## 1    20     1   0.3      1    49
## 2    20     1   0.3      1    50
```

2.2 Data generation function

`data_generation` allows users to flexibly change data while keeping the summary statistics and to choose the number of inputs by using different `purrr` mapping functions: map, map2, and pmap for a input, two inputs, and p inputs respectively.

In the function below, `simulation` means a distribution of data, and `grid` is a list of parameters.

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

  if(ncol(grid)==2){
    var1 <- c(unlist(grid[,1]))
    data <- map(var1, simulation)
    #different purrr-functions depending on how many input variables we use
```

```
  }

  if(ncol(grid)==3){
    var1 <- c(unlist(grid[,1]))
    var2 <- c(unlist(grid[,2]))
    data <- map2(var1, var2, simulation)
  }

  if(ncol(grid)==4){
    var1 <- c(unlist(grid[,1]))
    var2 <- c(unlist(grid[,2]))
    var3 <- c(unlist(grid[,3]))
    list1 <- list(var1,var2,var3)
    data <- pmap(list1, .f=simulation)
  }

  return(data)
}
```

`data_generation()` Example:

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

```
## $n6
##  [1] -0.491031166 -2.309168876  1.005738524 -0.709200763 -0.688008616
##  [6]  1.025571370 -0.284773007 -1.220717712  0.181303480 -0.138891362
## [11]  0.005764186  0.385280401 -0.370660032  0.644376549 -0.220486562
## [16]  0.331781964  1.096839013  0.435181491 -0.325931586  1.148807618
```

```
grid2 <- create_grid(param_list2, nrep=3)
tail(data_generation(simulation=rnorm, grid=grid2),1)
```

```
## $n30
##  [1]  1.9672673  0.8917199  0.3015793  0.7240548  2.1146485  1.5500440
##  [7]  2.2366758  1.1390979  1.4102751  0.4415431  1.6053707  0.4936665
## [13] -0.4205655  1.1279930  2.9458512  1.8009143  2.1652534  1.3588557
## [19]  0.3914428  0.7977591
```

Users can apply many distributions such as normal, uniform, poisson distributions by putting existing functions in r as `simulation`.

```r
# 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] 0.004638151 0.277560080 0.325203143 0.588706277 0.249684701 0.043117281
##  [7] 0.110678788 0.703753812 0.939021239 0.311169018 0.078492930 0.321744091
## [13] 0.624905537 0.440241850 0.801345301 0.279283805 0.570713193 0.042128012
## [19] 0.190717455 0.727086471 0.826690050 0.510721075 0.567726166 0.001155820
## [25] 0.143778103 0.865967083 0.082561061 0.244570682 0.981543157 0.577581279
```

```r
# 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
```

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

```
## $n99
##  [1]  8  8  8 12  7  6 10  9  5  8 19 12  7 12 13  7  3  9  7 15  6 13 11 15  8
## [26] 13  9  7  9  5
```

2.3 Summary function

`summary_function` offers summary statistics that users can choose.

```r
#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)
}
```

summary_function Example:

```r
grid_test <- create_grid(param_list3, nrep=3)
test_data <- data_generation(simulation=rnorm, grid=grid_test)
tail(summary_function(sum_fun=list("mean"), data_input=test_data),2)
```

```
##            mean
## [119,] 1.03361
## [120,] 1.01786
```

2.4 Summary array funcation

The outcome of `create_array_function` illustrates the combination of user defined grid and the summary statistics. This function product dataframes with all permutations and results that allow, thus users can look any possible parameter regarding specific grid.

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

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)
                     ,c("sd", 0, 1, 1))

comb1 <- main_function_array_test(parameters=param_list3x
                                  , nrep = 1
                                  , simulation = rnorm
                                  , sum_fun="mean")

head(comb1,2)
```

```
##    n mu sd rep       mean
```

```
## 1 10  0  0   1  0.0000000
## 2 10  0  1   1 -0.6031898
```

```
create_array_function(comb=comb1, parameters=param_list3x, nrep=1)
```

```
## , , 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 -0.6031898 1.1547493 1.768505 2.799209 4.297611 5.240045
## n=20 -0.1950611 0.7933902 1.609615 2.815089 4.066077 4.798390
```

3 Monte Carlo Simulation Funcion

4 Examples

5 Conclusion

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