SCIENTIFIC BULLETIN
OF
THE POLITEHNICA UNIVERSITY OF TIMISOARA

Vol.66(80)	MATHEMATICS–PHISYCS	2021

SOME APPLICATIONS OF MONTE CARLO METHOD IN MEDICINE

Theodora CARTIANU

Abstract

Simulation currently affects our daily lives through our interactions with the automotive industry, airlines, and entertainment. The use of simulation in drug development is relatively new, but its usage is increasing in line with the speed of modern computers. A well-known example of simulation in drug development is molecular modeling. Another use of simulation that has recently been observed in drug development is Monte Carlo simulation in clinical studies. Monte Carlo simulation differs from traditional simulation in that the model parameters are treated as stochastic or random variables rather than fixed values. The purpose of this paper is to provide a brief introduction to Monte Carlo simulation methods.

Computer simulation in the pharmaceutical industry is used in the discovery of new drugs, optimization of chemical processes, and most recently, the design of clinical trials. What most people consider "simulation" are models that build or design physical "things". Examples of this type of simulation include the use of computer-aided design (CAD) technology in designing a commercial product such as a car or an airplane, or molecular modeling of drug-receptor interactions. Often, with this class of simulations, the term "modeling" is used interchangeably, such as molecular modeling. For our purposes, models will be differentiated from simulations in that models are built upon data and look back in time, while simulations are based on models and look forward in time. [3]

1 Introduction

The purpose of this paper is not to review those simulations meant to mimic the real world and build physical things, but rather to provide a brief introduction to the class of simulations that

[3]Mathematical Subject Classification (2020): {*62-04, 62P10, 65C05*}
Keywords and phrases: Monte Carlo simulation method, modeling, clinical studies

BUPT

utilize mathematical models of a system, which may include physical processes, events, or decision-making processes, to describe the behavior of the system over long periods of time and incorporate stochastic or random variability into the model. This class of simulation is used for decision-making, understanding complex systems, and solving problems for which there is no closed-form solution. In 1979, the Society for Computer Simulation presented a framework for computer simulations (Fig. 1). In this framework, a problem from the real world is first identified and then converted into a conceptual model, which, for our purposes, will be defined as an abstraction of a real system that contains inputs and outputs, where the inputs of the model are predictive with the observed outputs. Ideally, the model is defined using a mechanistic basis and validated using observational data, but neither is necessarily required. Models have the advantage of allowing a better understanding of processes and systems, enabling the user to manipulate inputs to examine outputs.

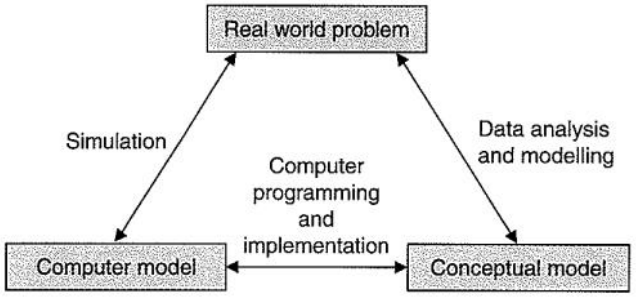


Fig. 1. Framework for computer simulation (modified by the Technical Committee of the Society for Computer Simulation on Model Credibility).

Source: Society for Computer Simulation (SCS) Technical Committee on Model Credibility. Terminology of model credibility. Simulation 1979 Mar: 103-4.

Models and, consequently, simulations can be further classified into discrete event simulations or real-time simulations. Discrete event simulations model a system over time, where variables change instantaneously at certain moments. An example of this is when the occurrence of an adverse reaction happens when plasma concentrations reach a threshold limit. Real-time simulations, which are often observed in clinical trial simulations, model systems where the system changes gradually over time. Of course, some simulations can be a combination of discrete events and real-time processes.

Monte Carlo simulation is the term applied to stochastic simulations, either discrete events, real-time simulations, or a combination thereof, incorporating random variability into the model. The term "Monte Carlo" was first coined by Ulam and von Neumann during World War II and referred to the games of chance in Monaco. The distribution is defined a priori (typically a normal distribution with mean and variance). The Monte Carlo method efficiently simulates the model by repeatedly sampling different random sets of values (inputs) from the parameter sampling distribution, resulting in a set of possible outcomes (outputs).

2 Implementation of Monte Carlo Method in R

Modeling and simulation of clinical studies is a tool used by pharmaceutical companies and the FDA to improve the efficiency of drug development. Monte Carlo Simulation is a modern and computationally efficient algorithm [1]. Therefore, it is a brilliant technique in terms of patient recruitment process and dose calculation in clinical design [2].

The purpose of this paper is to describe how Monte Carlo simulations have the task of evaluating

BUPT

parameter distributions and their applications in PROC MI.

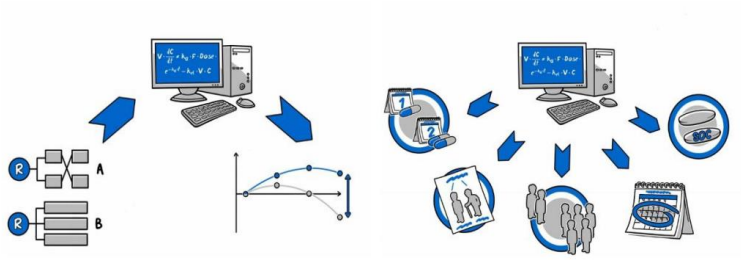
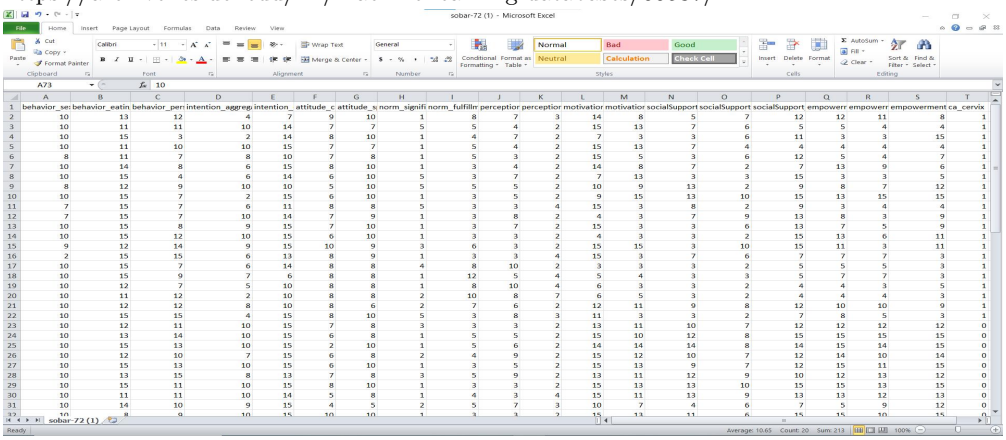


Fig. 2. Clinical Trial Simulation (CTS) explores how different studies design performance to detect the expected effect of the drug. Optimization includes dose regimen, patient profile, sample size, study duration, and the current standard of care (SOC) for patients in the clinical study simulation.

Source: Peter L. Bonate. Clinical Trial Simulation in Drug Development, 2000.

For the univariate data simulation in the data step, we will run R code with the loading of the dataset that can be found at:

https://archive.ics.uci.edu/ml/machine-learning-databases/00537/

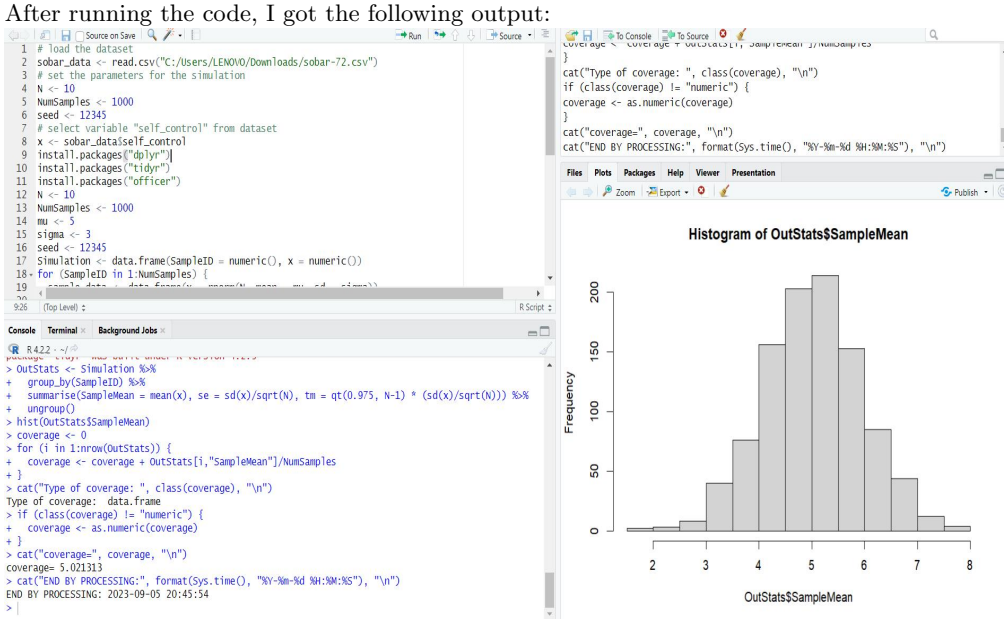


```
# load the dataset
sobar_data <- read.csv("C:/Users/LENOVO/Downloads/sobar-72.csv")
# set the parameters for the simulation
N <- 10
NumSamples <- 1000
seed <- 12345
# select variable "self_control" from dataset
x <- sobar_data$self_control
install.packages("dplyr")
install.packages("tidyr")
install.packages("officer")
N <- 10
NumSamples <- 1000
mu <- 5
sigma <- 3
seed <- 12345
Simulation <- data.frame(SampleID = numeric(), x = numeric())
```

BUPT

```
for (SampleID in 1:NumSamples) {
sample_data <- data.frame(x = rnorm(N, mean = mu, sd = sigma))
sample_data$SampleID <- SampleID
Simulation <- rbind(Simulation, sample_data)
}
library(dplyr)
library(tidyr)
OutStats <- Simulation %>%
group_by(SampleID) %>%
summarise(SampleMean = mean(x), se = sd(x)/sqrt(N), tm = qt(0.975, N-1) * (sd(x)/sqrt(N))) %>%
ungroup()
hist(OutStats$SampleMean)
coverage <- 0
for (i in 1:nrow(OutStats)) {
coverage <- coverage + OutStats[i,"SampleMean"]/NumSamples
}
cat("Type of coverage: ", class(coverage), "\n")
if (class(coverage) != "numeric") {
coverage <- as.numeric(coverage)
}
cat("coverage=", coverage, "\n")
cat("END BY PROCESSING:", format(Sys.time(), "%Y-%m-%d %H:%M:%S"), "\n")
```

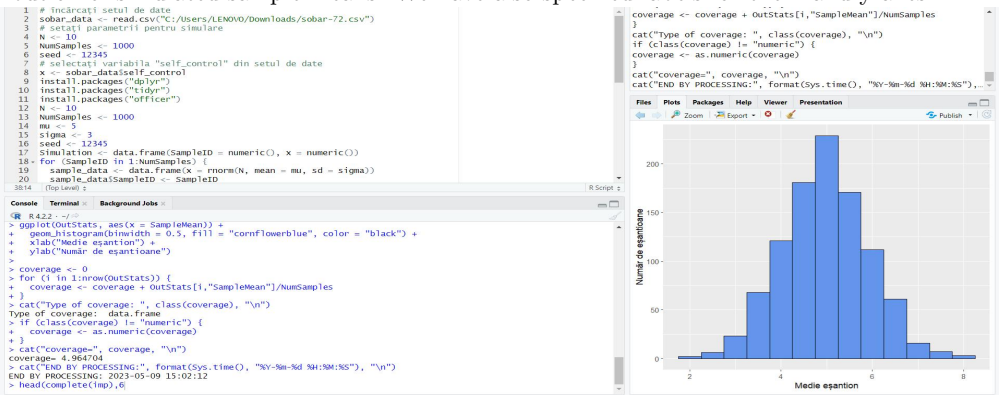
After running the code, I got the following output:



This code aims to perform a simulation of the distribution for the variable "self_control" from a dataset named "sobar-72.csv". Specifically, the code does the following: loads the dataset "sobar-72.csv" using 'read.csv'; sets simulation parameters such as the number of samples ('NumSamples') and the number of observations in each sample ('N'); generates a simulation of a normal distribution with specified mean ('mu') and standard deviation ('sigma') using the 'rnorm' function in a 'for' loop with 'NumSamples' iterations; calculates important statistics for each generated sample, including the mean ('SampleMean'), standard error ('se'), and upper quantile of the 95% confidence interval

('tm') using 'group_by', 'summarise', and 'ungroup' from the 'dplyr' package; plots a histogram of the sample means ('SampleMean'); calculates the coverage of the 95% confidence interval for the distribution of sample means and displays it using the 'cat' function; displays the end time of the process using 'format(Sys.time(), "%Y-%m-%d %H:%M:%S")'.

We obtained a coverage value of approximately 5, which indicates that the 95% confidence interval for the sample mean covers the true mean value with a probability of approximately 95% for each sample. Additionally, we can add the ggplot2 library and create a histogram plot for the distribution of simulated sample means. We have also specified labels for the x and y axes.



Display the histogram of the sample means distribution
ggplot(OutStats, aes(x = SampleMean)) +
geom_histogram(binwidth = 0.5, fill = "cornflowerblue", color = "black") +
xlab("Sample Mean") +
ylab("Number of Samples")

Furthermore, I have created a code that creates a Shiny application that allows the user to simulate data based on the chosen distribution and analyze the sensitivity of the simulated results. The user can select the distribution, sample size, and the mu and sigma parameters, and then can view the visualization and results in an interactive table. Furthermore, I have created a code that creates a Shiny application that allows the user to simulate data based on the chosen distribution and analyze the sensitivity of the simulated results. The user can select the distribution, sample size, and the mu and sigma parameters, and then can view the visualization and results in an interactive table.

# Install and load required packages
packages <- c("dplyr", "tidyr", "ggplot2", "shiny", "DT", "rmarkdown")
install.packages(packages, dependencies = TRUE)
sapply(packages, library, character.only = TRUE)
# load the dataset
sobar_data <- read.csv("C:/Users/LENOVO/Downloads/sobar-72.csv")
# Define variables for mu and sigma
mu <- 5
sigma <- 3
# set the parameters for the simulation
N <- 10
NumSamples <- 1000
seed <- 12345
# select variable "self_control" from dataset
x <- sobar_data$self_control
# Simulate data with the option to choose the distribution
simulate_data <- function(N, NumSamples, mu, sigma, distribution) {

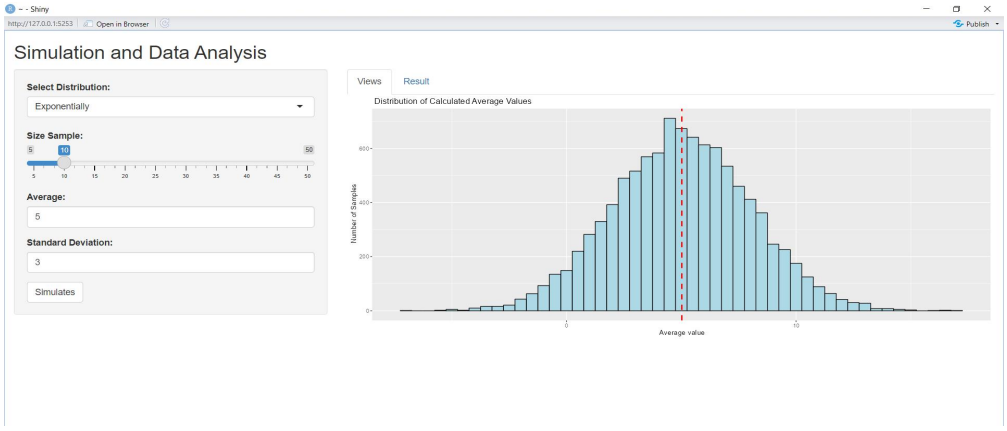
BUPT

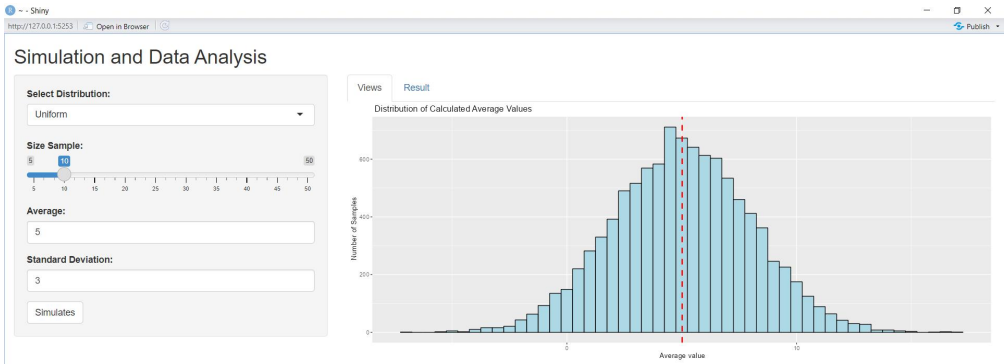
```
    Simulation <- data.frame(SampleID = numeric(), x = numeric())
    for (SampleID in 1:NumSamples) {
    if (distribution == "normal") {
    sample_data <- data.frame(x = rnorm(N, mean = mu, sd = sigma))
    } else if (distribution == "exponentially") {
    sample_data <- data.frame(x = rexp(N, rate = 1/mu))
    } else if (distribution == "uniform") {
    sample_data <- data.frame(x = runif(N, min = mu - sigma, max = mu + sigma))
    }
    sample_data$SampleID <- SampleID
    Simulation <- rbind(Simulation, sample_data)
    }
    return(Simulation)
    }
    # Data validation
    validate_data <- function(data) {
    # We can add checks for missing data or outliers here
    # For example: if (any(is.na(data$x))) { ... }
    return(data)
    }
    # Sensitivity analysis
    sensitivity_analysis <- function(N, NumSamples, mu, sigma, distribution) {
    results <- list()
    for (i in 1:10) { # Performs the analysis for 10 sets of parameters
    simulated_data <- simulate_data(N, NumSamples, mu, sigma, distribution)
    validated_data <- validate_data(simulated_data)
    OutStats <- validated_data %>%
    group_by(SampleID) %>%
    summarise(SampleMean = mean(x), se = sd(x)/sqrt(N), tm = qt(0.975, N-1) * (sd(x)/sqrt(N)))
%>%
    ungroup()
    coverage <- sum(OutStats$SampleMean)/NumSamples
    results[[i]] <- list(params = list(N = N, NumSamples = NumSamples, mu = mu, sigma =
sigma),
    coverage = coverage)
    }
    return(results)
    }
    # UI for Shiny application
    ui <- fluidPage(
    titlePanel("Simulation and Data Analysis "),
    sidebarLayout(
    sidebarPanel(
    selectInput("distribution", " Select Distribution:",
    choices = c("Normal" = "normal", "Exponentially" = "exponential", "Uniform" = "uni-
form")),
    sliderInput("N", " Size Sample:", min = 5, max = 50, value = N),
    numericInput("mu", " Average:", value = mu),
    numericInput("sigma", " Standard Deviation:", value = sigma),
    actionButton("simulateButton", " Simulates")
    ),
```

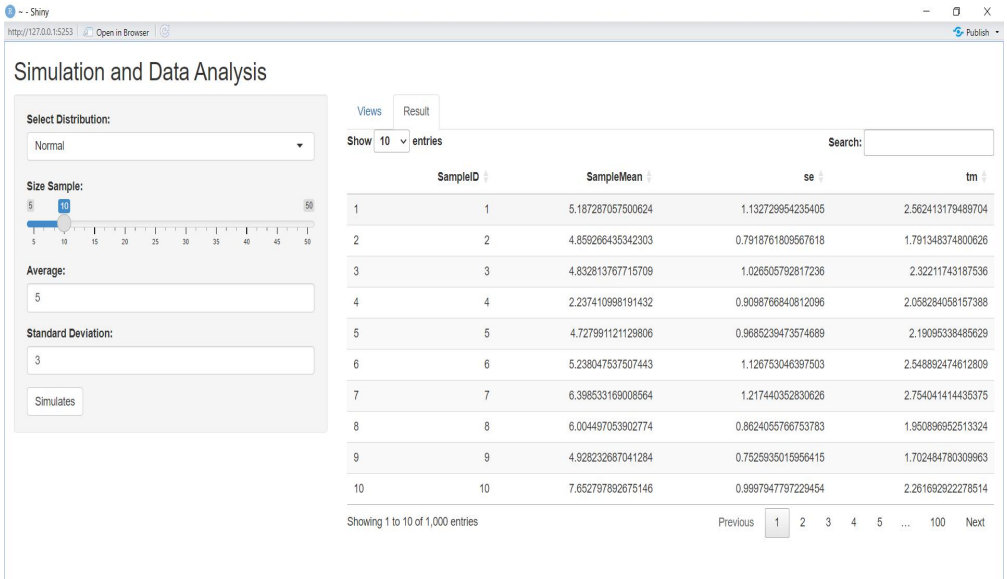
BUPT

```
mainPanel(
tabsetPanel(
tabPanel("Views ", plotOutput("histPlot")),
tabPanel("Result", dataTableOutput("summaryTable"))
)
)
)
)
# Server for Shiny application
server <- function(input, output) {
data <- eventReactive(input$simulateButton, {
simulate_data(input$N, NumSamples, input$mu, input$sigma, input$distribution)
})
output$histPlot <- renderPlot({
ggplot(data(), aes(x = x)) +
geom_histogram(binwidth = 0.5, fill = "lightblue", color = "black") +
geom_vline(aes(xintercept = mean(x)), color = "red", linetype = "dashed", size = 1) +
labs(title = " Distribution of Calculated Average Values",
x = " Average value",
y = " Number of Samples")
})
output$summaryTable <- renderDataTable({
OutStats <- data() %>%
group_by(SampleID) %>%
summarise(SampleMean = mean(x), se = sd(x)/sqrt(N), tm = qt(0.975, N-1) * (sd(x)/sqrt(N)))
%>%
ungroup()
datatable(OutStats)
})
}
# Run the Shiny application
shinyApp(ui = ui, server = server)
```







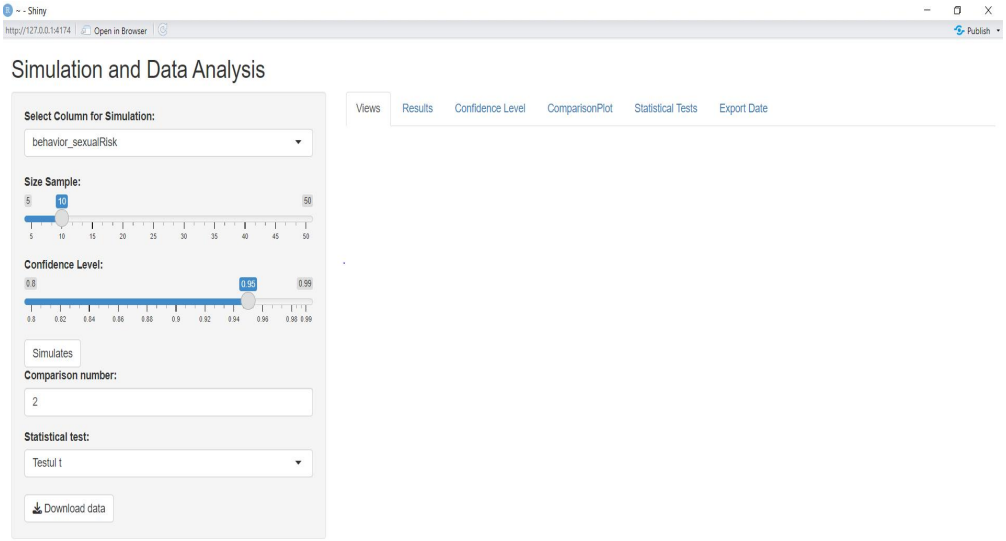


In addition, we can modify the code for the Shiny application in R, following these steps:

1. Installation and loading of necessary packages.

BUPT

2. Loading a dataset from the "sobar-72.csv" file and displaying the data.
3. Defining parameters for the simulation, such as mean, standard deviation, and sample size.
4. Simulating data based on the selected column from the dataset.
5. Creating a Shiny interface to allow the user to configure the simulation.
6. Defining a Shiny server that generates visualizations and results, including a histogram, result table, and confidence interval.
7. The ability to compare distributions and perform statistical tests (requires additional implementation).
8. Running the Shiny application for interactive exploration of simulations and results.



3 Missing Data Imputation using The MI Procedure

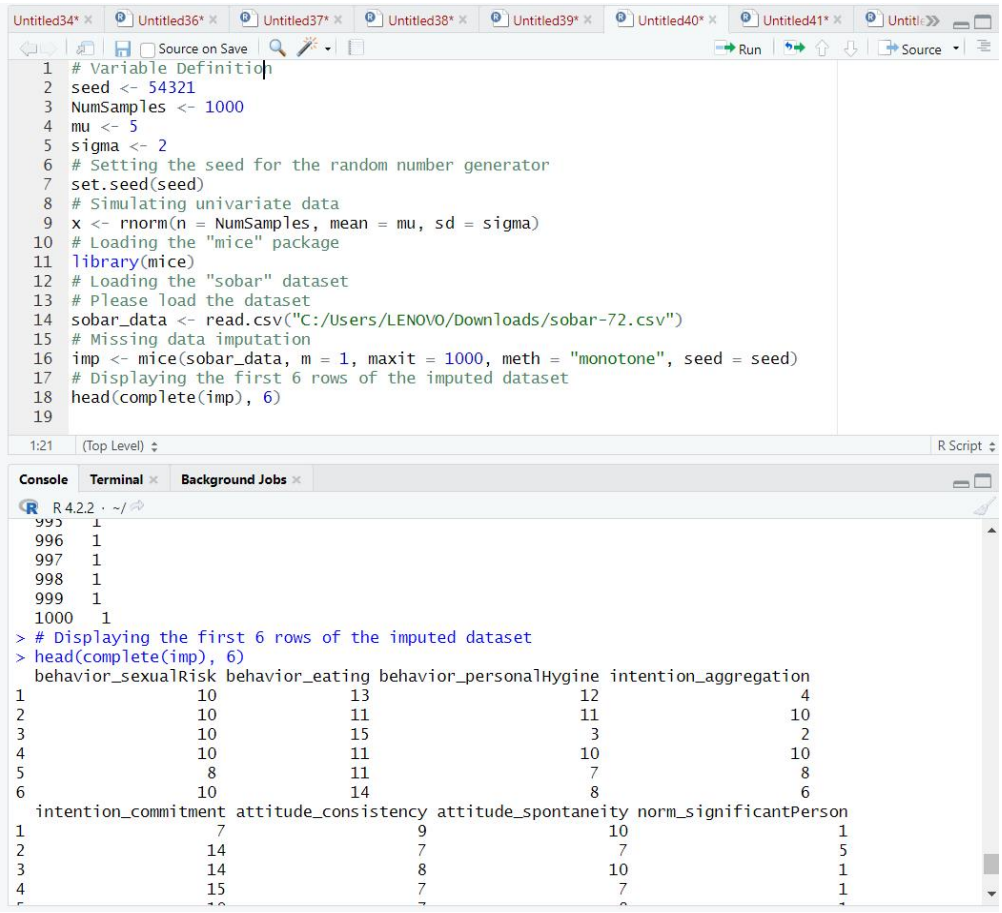
Missing values pose a problem in a substantial number of clinical analysis studies. Some subjects drop out of the study. Some data is missing due to the patient's illness or death, invalid measurements, or forgetfulness[3]. A statistical analysis can be biased if incomplete cases are excluded from the analysis due to the intent-to-treat (ITT) principle. The MI procedure employs the MCMC method for imputing missing data, simulating the parameters of the model, and diagnosing the model[4].

We will run R code with the loading of the dataset, which can be found at:
https://archive.ics.uci.edu/ml/machine-learning-databases/00537/

```
# Variable Definition
seed <- 54321
NumSamples <- 1000
mu <- 5
sigma <- 2
# Setting the seed for the random number generator
set.seed(seed)
# Simulating univariate data
x <- rnorm(n = NumSamples, mean = mu, sd = sigma)
# Loading the "mice" package
library(mice)
```

BUPT

```
# Loading the "sobar" dataset
# Please load the dataset
sobar_data <- read.csv("C:/Users/LENOVO/Downloads/sobar-72.csv")
# Missing data imputation
imp <- mice(sobar_data, m = 1, maxit = 1000, meth = "monotone", seed = seed)
# Displaying the first 6 rows of the imputed dataset
head(complete(imp), 6)
```

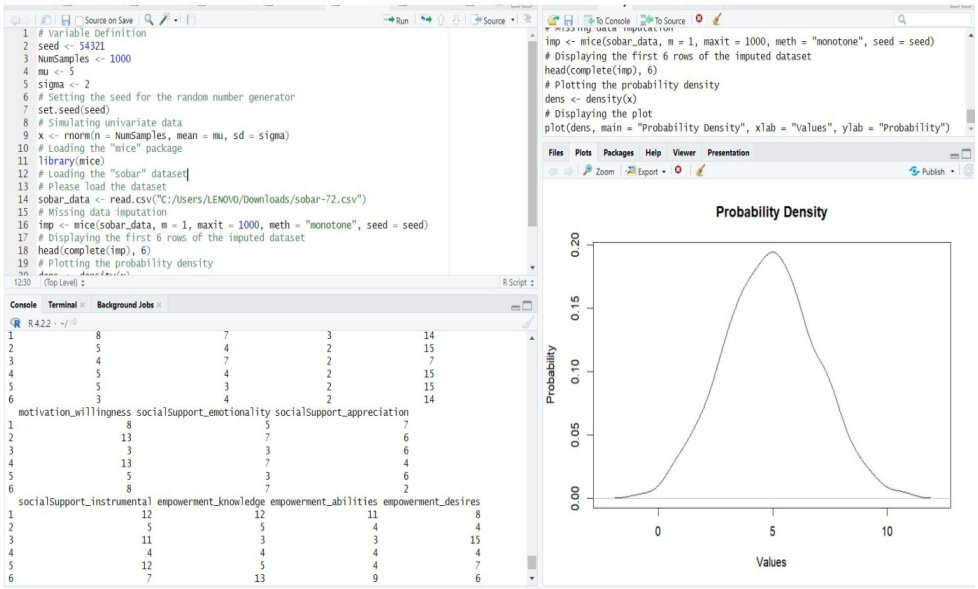


Following the execution of the code, the result will display the first 6 rows of the 'sobar_data' dataset with the missing values imputed using the "monotone" method with the help of the 'mice' package. This is achieved by applying the 'complete()' function to the 'imp' variable, which contains the dataset filled with imputed values.

Additionally, in the above code, we can add plotting the probability density as follows:

```
# Plotting the probability density
dens <- density(x)
# Displaying the plot
plot(dens, main = "Probability Density", xlab = "Values", ylab = "Probability")
```

BUPT



The displayed plot presents the probability density of the values in the ‘x‘ vector. The horizontal axis represents the values in the ‘x‘ vector, while the vertical axis represents the probability density for those values. In this case, the plot shows that the probability density is highest around the value 5, indicating that the values in the ‘x‘ vector are more likely to be closer to this value. Additionally, the probability density is lower around the extreme values, indicating that extreme values are less likely to occur in the ‘x‘ vector.

This code will perform a t-test to check if the mean of the data in x is different from 5.0 and will display the test results along with an interpretation based on the obtained p-value. We can adjust the value of mu and the significance level (0.05 in this example) as needed.

Hypothesis test for the mean of the data in x

hyp_test <- t.test(x, mu = 5)

Displaying the test results

cat("Hypothesis test results:\n")

print(hyp_test)

Displaying the interpretation of the results

cat("\nInterpretation of the results:\n")

if (hyp_test$p.value < 0.05) {

cat("The p-value is less than 0.05, so we reject the null hypothesis.\n")

cat("The mean of the data in x is different from 5.0.\n")

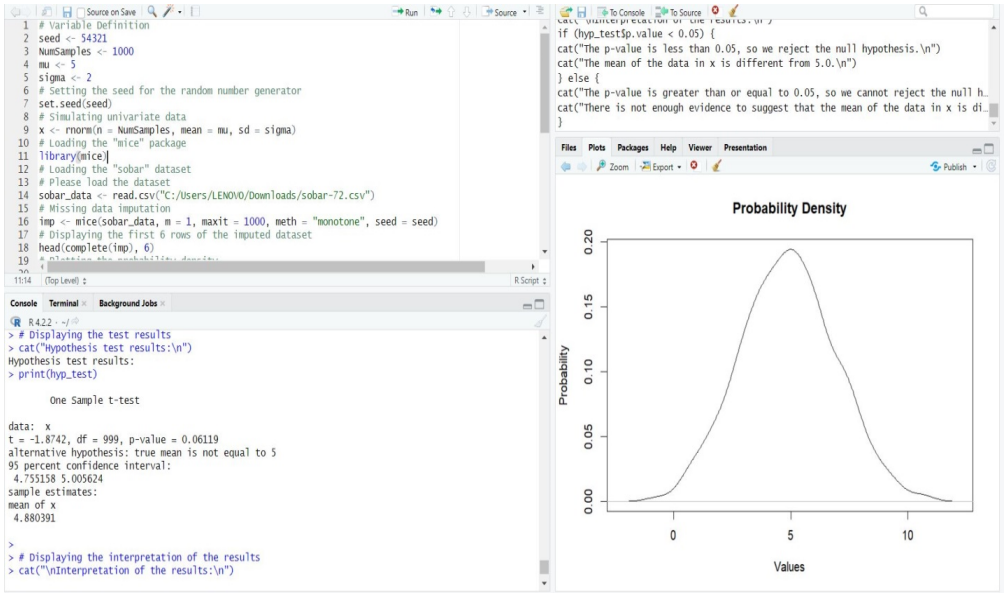
} else {

cat("The p-value is greater than or equal to 0.05, so we cannot reject the null hypothesis.\n")

cat("There is not enough evidence to suggest that the mean of the data in x is different from 5.0.\n")

}

BUPT




```
mean of x
 4.880391

>
> # Displaying the interpretation of the results
> cat("\nInterpretation of the results:\n")

Interpretation of the results:
> if (hyp_test$p.value < 0.05) {
+   cat("The p-value is less than 0.05, so we reject the null hypothesis.\n")
+   cat("The mean of the data in x is different from 5.0.\n")
+ } else {
+   cat("The p-value is greater than or equal to 0.05, so we cannot reject the null hypothesis.\n")
+   cat("There is not enough evidence to suggest that the mean of the data in x is different from 5.
0.\n")
+ }
The p-value is greater than or equal to 0.05, so we cannot reject the null hypothesis.
There is not enough evidence to suggest that the mean of the data in x is different from 5.0.
>
```

4 Conclusions

The Monte Carlo method is a technique used to simulate the behavior of complex systems by generating repeated random values. This method can be used in medicine to estimate the probability of success of a medical procedure, optimize the design and dosage of medications, or evaluate the impact of a new treatment on patients.

Regarding applications in medicine, the Monte Carlo method can be used to simulate treatments and their effects on patients, allowing physicians to better understand the risks and benefits of a particular treatment before administering it to patients. Additionally, this method can be used in the design and testing of new medical devices, such as implants or diagnostic equipment.

Although the Monte Carlo method is very powerful and useful in medicine, there are also some challenges and limitations in its use. For example, in the case of simulating treatments and medical procedures, it can be influenced by individual variations among patients and complex interactions between different elements of the system.

Moreover, solid programming and mathematical modeling expertise are required to successfully implement the Monte Carlo method in medical research.

BUPT

Therefore, it is important for researchers and physicians to consider these limitations and collaborate with experts in the field of statistical methods and mathematical modeling to achieve the best possible outcomes.

The Monte Carlo method is a powerful and versatile tool in the field of medicine, which can help optimize processes and treatments, reduce risks, and improve the quality of patient care.

However, it is important to consider the limitations of this method and combine it with other methods and technologies to ensure the best possible solution for each specific medical situation.

In conclusion, I would like to thank my supervisor, Associate Professor Dr. Negrea Romeo, as well as my family for all the support they have provided.

References

[1] Munsaka, Melvin, Using SAS for Modeling and Simulation in Drug Development – A Review and Assessment of Some Available Tools, 2011.

bibitem2 Bonate, Peter L, Clinical Trial Simulation in Drug Development, 2000.

[2] Wicklin, Rick, SIMULATING DATA with SAS, SAS online document, 2013.

[3] Patterson, Scott D., SAS Markov Chain Monte Carlo (MCMC) Simulation in Practice, 2007.

vspace-6pt

Theodora CARTIANU
Master student MTSSCC, Politehnica University of Timisoara
Adresa: P-ta victorie 2, 300006, Timisoara, Romania
E-mail:theodora.cartianu@yahoo.com

BUPT