1. Exercise 1. Topic 1: Pseudo-random number generator „Randu”.

1.1. Implement a RANDU generator.

The formula for RANDU is explained in (1) as stated in Equation 1, where x_i is the i-th value in the sequence.

$$x_i = 65539 \times x_{i-1} \mod 2^{31} \quad (1)$$

The implementation in R is the following:

```
##### Part 1.1: Implement a RANDU generator
generate_randu <- function(M) {
  a <- 65539 # multiplier
  m <- 2^31 # modulo
  x <- numeric(M)
  x[1] <- 1 # seed = 1 as requested in the exercise
  for (i in 2:M) {
    x[i] <- (a * x[i-1]) %% m # RANDU formula
  }
  u <- x / m
  return(u)
}
```

1.2. Illustrate RANDU in 2D.

The exercise asks to mimic the figure presented in (2). For this reason, the following code has been implemented:

```
##### Part 1.2: Illustrate RANDU in 2D

png("exercise_1_2_RANDU_2D.png")
plot(randu_numbers[1:(M-1)], randu_numbers[2:M],
     main = "RANDU in two dimensions", # Title of the plot
     xlab = expression(u[i-1]), ylab = expression(u[i]), # Label of the axis as
in the lectures
     pch = 19, col = "blue", cex = 0.1) # setup of the points
dev.off()
```

The result of the code is presented in Figure 1.

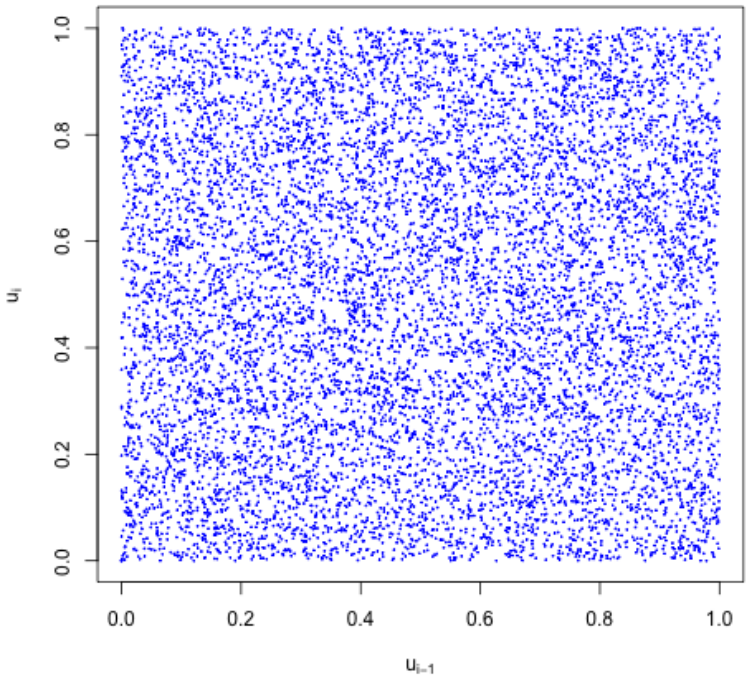


Figure 1: Randu in two dimensions

1.3. Randu in three dimensions.

The third part of the exercise consists of showing the RANDU in three dimensions, mimicking the behavior (3) shown in the lecture. To do so, an interactive graph has been implemented as follows:

```
plot3d(randu_numbers[1:(M-2)], randu_numbers[2:(M-1)], randu_numbers[3:M],
      col = "blue", size = 3, xlab = expression(u[i]), ylab = expression(u[i-1]),
zlab = expression(u[i-2]))
```

The result is shown in Figure 2.

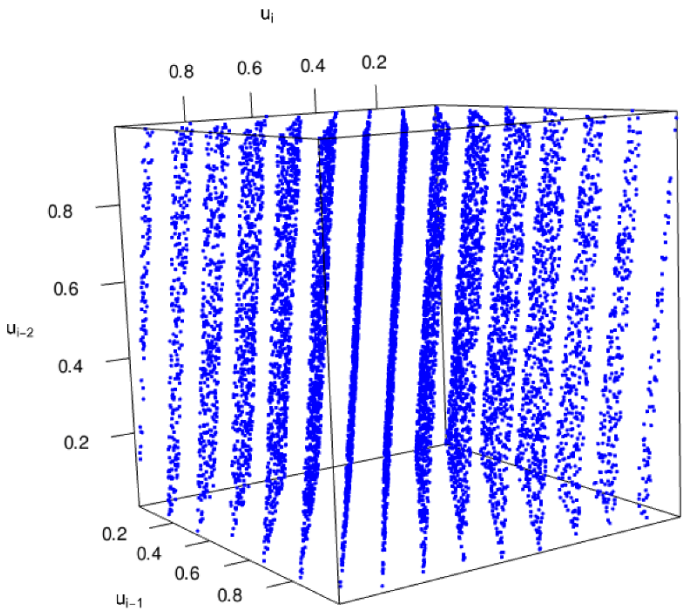


Figure 2: Randu in three dimensions

1.4. Approximate π.

The last task of the first exercise asks to approximate the π. To do so, the method presented in the last week has been used (4) and the following function has been implemented:

```
##### Part 1.4: Approximate the number pi

approximate_pi <- function(u) {
  M <- length(u) / 2 # Half of the generated numbers
  x <- u[1:M] # Taking the first half
  y <- u[(M+1):(2*M)] # Taking the second half
  inside_circle <- sum(x^2 + y^2 <= 1) # Evaluate how many points are inside
  pi_approx <- (inside_circle / M) * 4 # Calculate pi
  return(pi_approx)
}
```

The result of the approximation leads to 3.1496. Additionally, the exercise requests to produce an R-plot to illustrate the approach (cf., Figure 3). The image illustrates a quarter of a circle: in blue, the points inside the circle, and in red, the points outside the circle. The code for the plot generation is also provided.

```
png("exercise_1_4_pi_approximation.png")
plot(x, y, col = ifelse(inside_circle, "blue", "red"),
     main = "Monte Carlo Approximation of Pi",
     xlab = "x", ylab = "y", pch = 19)
```

```
legend("topright", legend = c("Inside Circle", "Outside Circle"),
       col = c("blue", "red"), pch = 19)
dev.off()
```

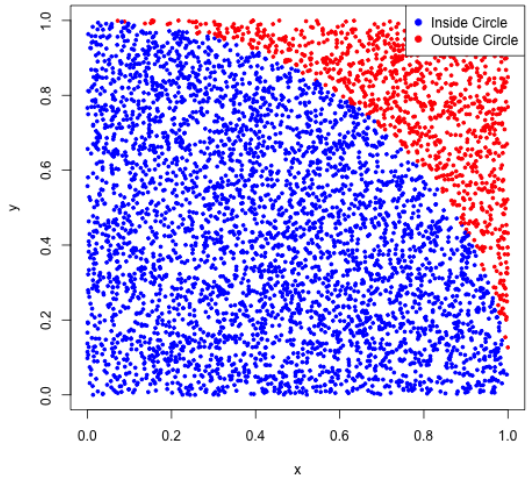


Figure 3: Monte Carlo approximation of π

2. Exercise 2. Topic 2: Acceptance-rejection method and Box-Muller algorithm

First of all, I defined the function $c(x)$ as:

$$c(x) = \frac{\frac{2}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}}{e^{-x}} = \frac{2}{\sqrt{2\pi}} \cdot e^{x-\frac{x^2}{2}}$$

To find the maximum value of $c(x)$, I set the derivative to zero. Before that, I used the function $h(x) = x - \frac{x^2}{2}$ which corresponds to the exponent of e. Since the derivative of $c(x)$, using the derivative chain rule, will be a product w.r.t. $h'(x)$, it would be enough to set $h'(x) = 0$ to find the critical point. In this case, the critical point occurs at $x = 1$.

Therefore, the $c(1)$ will be:

$$c(1) = \frac{2}{\sqrt{2\pi}} \cdot e^{1-\frac{1^2}{2}} = \frac{2}{\sqrt{2\pi}} \cdot e^{0.5} = \frac{\sqrt{2} \cdot \sqrt{2}}{\sqrt{2} \cdot \sqrt{\pi}} \cdot \sqrt{e} = \sqrt{\frac{2e}{\pi}}$$

After finding the maximum value for the ratio, I defined the functions in R as:

```
# Define the target probability density function
f <- function(x) {
  (2 / sqrt(2 * pi)) * exp(-x^2 / 2)
}

# Define the exercise density function g(x)
```

```r
g <- function(x) {
  exp(-x)
}

# Determine the constant c as the maximum of f(x) / g(x)
c <- sqrt(2 * exp(1) / pi)
```

2.1. Implement acceptance-rejection.

After that, I implemented the acceptance-rejection method explained in the lecture (5) as:

```r
# Perform acceptance-rejection
while (count < M) {
  Y <- rexp(1)  # Sample from the proposal distribution g(x)
  U <- runif(1)  # Sample from uniform random number 0-1
  total_trials <- total_trials + 1  # Increment the total number of trials
  # Check if it is less or equal than the acceptance ratio
  if (U <= f(Y) / (c * g(Y))) {
    count <- count + 1 # Counter for accepted samples
    samples[count] <- Y # Store accepted sample
  }
}

# Display the first few accepted samples
head(samples)
```

The first few accepted samples are: [0.8434573, 1.3290549, 0.1028700, 0.3142273, 1.4154478, 0.3856068].

2.2. Determine the acceptance probability.

For computing the accepting probability, I first computed the theoretical probability as $\frac{1}{c} \approx 0.7601735$ (6) and then the observed probability as the number of accepted realizations divided by the total trials of samples. The result of the following code is 0.7634753.

```r
# Theoretical acceptance probability
acceptance_probability_theoretical <- 1 / c
print(acceptance_probability_theoretical)
# 0.7601735

# Observed acceptance probability
acceptance_probability_observed <- M / total_trials
print(acceptance_probability_observed)
# 0.7634753
```

2.3. Implement the Box-Müller algorithm.

To implement the Box-Muller algorithm presented in the lecture notes (7), the following functions have been implemented:

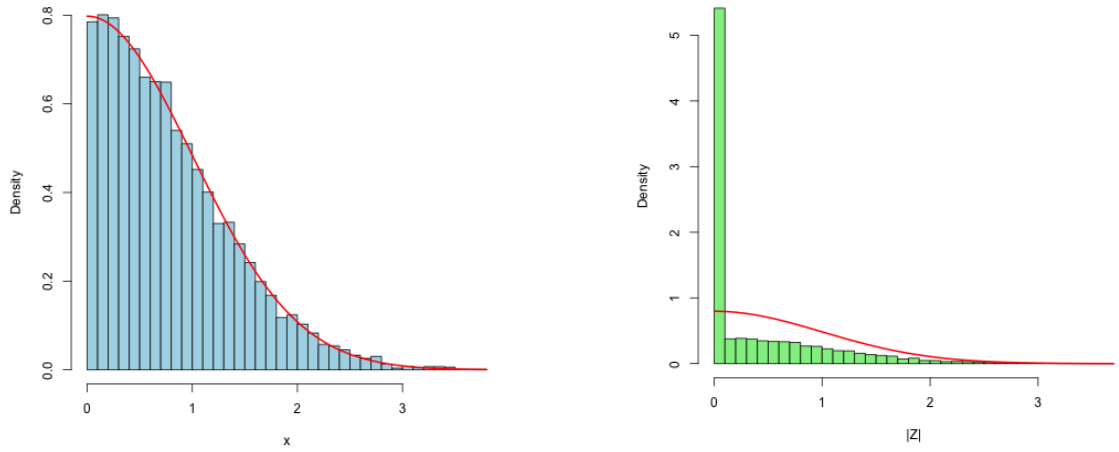
```
# Generating normal samples using Box-Muller transform
for (i in 1:(M/2)) {
  U1 <- runif(1) # Generate first uniform random number
  U2 <- runif(1) # Generate second uniform random number

  # Apply Box-Muller transform
  Z1[i] <- sqrt(-2 * log(U1)) * cos(2 * pi * U2)
  Z2[i] <- sqrt(-2 * log(U1)) * sin(2 * pi * U2)
}
```

A few algorithm results are [0.3763186, 0.9919799, 0.3361111, 0.8815332, -1.0507947, -0.2843938].

2.4. Compare empirical pdfs.

Figure 4 shows the two empirical pdfs for Acceptance-Rejection (a) and Box-Muller (b).



(a) (b)

Figure 4: PDF for Acceptance-Rejection (a) and Box-Muller (b)

3. Exercise 3. Topic 3: Markov chain

3.1. Produce a corresponding transition diagram.

Figure 5 shows the exercise transition diagram, as explained during the lectures (8), obtained with the following code:

```
# Define transition diagram
graph <- grViz("
digraph markov_chain {
  rankdir=LR;
```

```
  node [shape = circle];
  1 [label = 'non-infected'];
  2 [label = 'infected'];
  3 [label = 'hospitalized'];
  4 [label = 'intensive care unit'];
  5 [label = 'dead'];

  1 -> 1 [label = '0.93'];
  1 -> 2 [label = '0.07'];

  2 -> 1 [label = '0.05'];
  2 -> 2 [label = '0.80'];
  2 -> 3 [label = '0.10'];
  2 -> 4 [label = '0.05'];

  3 -> 2 [label = '0.15'];
  3 -> 3 [label = '0.80'];
  3 -> 4 [label = '0.05'];

  4 -> 3 [label = '0.05'];
  4 -> 4 [label = '0.80'];
  4 -> 5 [label = '0.15'];

  5 -> 5 [label = '1'];
}
")

# Save the diagram as SVG
svg_filename <- "exercise_3_1_transition_diagram.svg"
svg_content <- export_svg(graph)

# Save the SVG content to a file
write(svg_content, file = svg_filename)

# Convert the SVG to PNG
png_filename <- "exercise_3_1_transition_diagram.png"
rsvg_png(svg_filename, png_filename)
```

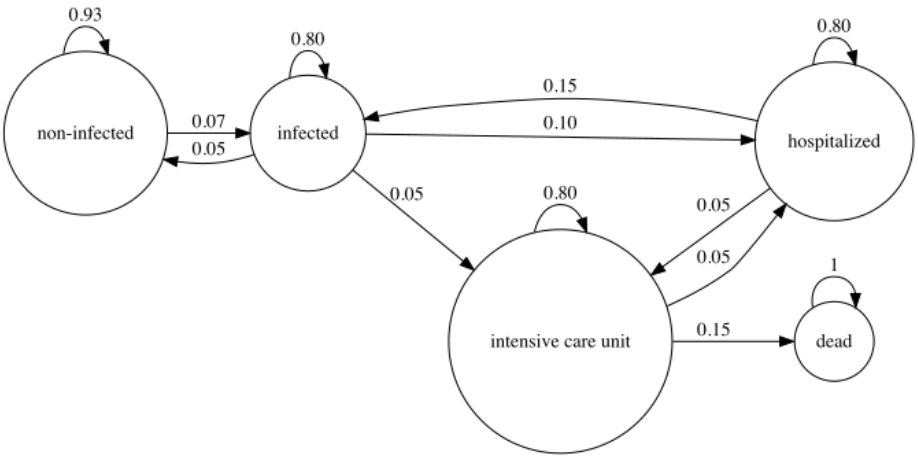


Figure 5: Transition diagram

3.2. Implement an algorithm to generate a path of the Markov chain.

The following code represents my implementation of generating the Markov chain. It is inspired by the code presented in the slides (9).

```
# Generate the path of the Markov chain
generate_path <- function(P, X0, N) {
  path <- numeric(N + 1) # Initialize vector
  path[1] <- X0 # Set initial condition
  for (i in 2:(N + 1)) {
    path[i] <- sample(1:5, size = 1, prob = P[path[i - 1], ]) # sample next state
  }
  return(path)
}
```

3.3. Plot the result for $X_0 = 1$ and $N = 30$ time steps.

Figure 6 represents the requested plot for the given conditions:

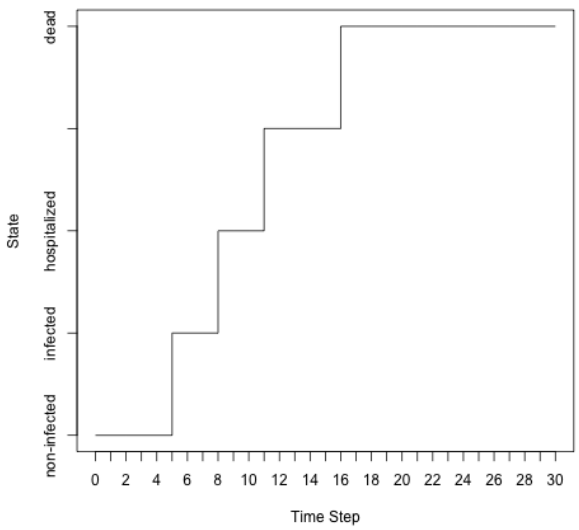


Figure 6: Markov chain path for initial state 1, and 30 time steps

3.4. Consider random variable τ.

To solve this task, I implemented the following code:

```
# Estimate the life expectancy
estimate_life_expectancy <- function(P, X0, M) {
  tau <- numeric(M) # Initialize the vector for the time to absorption
  for (i in 1:M) {
    n <- 0 # Initialize time counter
    state <- X0 # Set initial state
    while (state != 5) {  # Until absorbing state (dead) is reached
      state <- sample(1:5, size = 1, prob = P[state, ]) # Sample the next state
      n <- n + 1 # Increment time counter
    }
    tau[i] <- n
  }
  return(mean(tau))
}

M <- 10^4  # Number of realizations
life_expectancy <- numeric(4) # Initialize vector for life expectancy
for (i in 1:4) {
  life_expectancy[i] <- estimate_life_expectancy(P, i, M)
}
```

3.5. Comparison between simulated and theoretical values.

For this task is requested to construct a matrix Q and compare the results with the values obtained in Section 3.4. The results of the comparison are stated in the Table 1.

State	Simulated	Theoretical
Non-infected	59.4481	59.52381
Infected	45.0570	45.23810
Hospitalized	42.8106	42.85714
Intensive care unit	16.2120	15.71429

Table 1: Comparison of F1

4. Exercise 4. Topic 4: Numerical Illustration of the Black-Scholes Model

The first three tasks deal with re-writing the explicit solution.

1. For $k \in \{0, \dots, N-1\}$ let's write in terms S_{t_k} at time t_k and the increment $\nabla W_{k+1} = W_{t_{k+1}} - W_{t_k}$ the $S_{t_{k+1}}$ at time t_{k+1} (where $\Delta t = t_{k+1} - t_k$:

$$S_{t_{k+1}} = S_{t_k} \exp\left(\left(r - \frac{\sigma^2}{2}\right)\Delta t + \sigma W_{k+1}\right)$$

2. Now let's write in terms of $S^e_{t_k}$ the Euler scheme at time t_k and the increment $\Delta W_{k+1} = W_{t_{k+1}} - W_{t_k}$ the approximation $S^e_{t_{k+1}}$ of the Euler scheme at time t_{k+1}:

$$S^e_{t_{k+1}} = S^e_{t_k} + rS^e_{t_k}\Delta t + \sigma S^e_{t_k}\Delta W_{k+1}$$

3. Similarly, I can write in terms of $S^m_{t_k}$ the Milstein scheme at time t_k and the increment $\nabla W_{k+1} = W_{t_{k+1}} - W_{t_k}$ the approximation $S^m_{t_{k+1}}$ of the Milstein scheme at time t_{k+1}:

$$S^m_{t_{k+1}} = S^m_{t_k} + rS^m_{t_k}\Delta t + \sigma S^m_{t_k}\Delta W_{k+1} + \frac{1}{2}\sigma^2 S^m_{t_k}((\Delta W_{k+1})^2 - \Delta t)$$

4.4. Implement the trajectories.

The following functions have been defined to define the trajectories:

```
# Part 4.1: Define the exact solution for the Black-Scholes model
exact_solution <- function(s0, t, Wt) {
  s0 * exp((r - 0.5 * sigma^2) * t + sigma * Wt)
}

# Part 4.2: Implement the Euler scheme
euler_scheme <- function(S, dt, dW) {
  S + r * S * dt + sigma * S * dW
}

# Part 4.3: Implement the Milstein scheme
milstein_scheme <- function(S, dt, dW) {
  S + r * S * dt + sigma * S * dW + 0.5 * sigma^2 * S * (dW^2 - dt)
}
```

Additionally, Figure 7 compares the trajectories resulting from the same increments as requested in the exercise (see (10)).

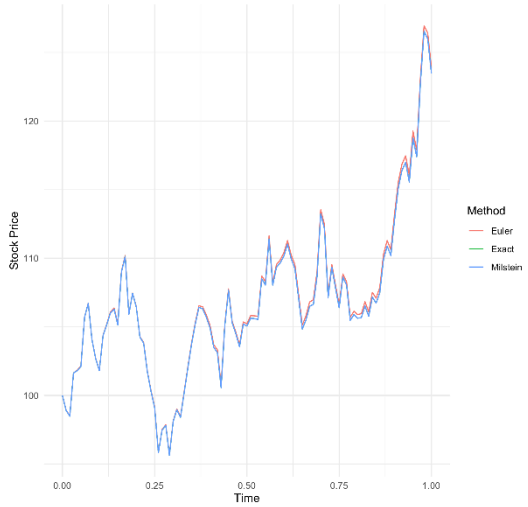


Figure 7: Black-Scholes Model comparison between Exact, Euler and Milstein Schemes

4.5. Calculate the empirical mean and L2 error.

The theoretical behavior of $\mathbb{E}((S_T - S_T^e)^2)$ as a function of N, the number of time steps, can be understood by analyzing the convergence of numerical schemes used to approximate solutions to SDEs. As N increases (and consequently Δt decreases) the approximation error $\mathbb{E}((S_T - S_T^e)^2)$ should decrease.

4.6. Confidence interval for the empirical mean

They are explained in the next section.

4.7. Evolution of the empirical mean and confidence intervals

With the following code, it is possible to see the empirical mean and the confidence intervals. For the Euler, the mean and confidence intervals result to be 865.1899 and [780.6226, 949.7572], respectively, while for the Milstein, the mean is 916.3187, and the confidence interval is [820.5745, 1012.063]. Figure 8 represents the evolution varying the M.

```
S_T_exact <- numeric(M)
S_T_euler <- numeric(M)
S_T_milstein <- numeric(M)

for (i in 1:M) {
  S_T_exact[i] <- simulate_trajectory(exact_solution, s0, dt, N)
  S_T_euler[i] <- simulate_trajectory(euler_scheme, s0, dt, N)
  S_T_milstein[i] <- simulate_trajectory(milstein_scheme, s0, dt, N)
}

# Calculate the empirical mean and L^2 error
```

```r
empirical_mean_euler <- mean((S_T_exact - S_T_euler)^2)
empirical_mean_milstein <- mean((S_T_exact - S_T_milstein)^2)

# Print the results
cat("Empirical mean (Euler):", empirical_mean_euler, "\n")
# Empirical mean (Euler): 865.1899
cat("Empirical mean (Milstein):", empirical_mean_milstein, "\n")
# Empirical mean (Milstein): 916.3187

### Part 4.6: Confidence interval for the empirical mean
alpha <- 0.05  # significance level
z_alpha <- qnorm(1 - alpha / 2) # Z-value for the given significance level

# Calculate confidence intervals for the empirical means
SE_euler <- sqrt(var((S_T_exact - S_T_euler)^2) / M)
CI_euler <- c(empirical_mean_euler - z_alpha * SE_euler, empirical_mean_euler +
z_alpha * SE_euler)

SE_milstein <- sqrt(var((S_T_exact - S_T_milstein)^2) / M)
CI_milstein <- c(empirical_mean_milstein - z_alpha * SE_milstein,
empirical_mean_milstein + z_alpha * SE_milstein)

# Print the confidence intervals
cat("Confidence interval (Euler): [", CI_euler[1], ", ", CI_euler[2], "]\n")
# Confidence interval (Euler): [ 780.6226 ,  949.7572 ]
cat("Confidence interval (Milstein): [", CI_milstein[1], ", ", CI_milstein[2],
"]\n")
# Confidence interval (Milstein): [ 820.5745 ,  1012.063 ]
```

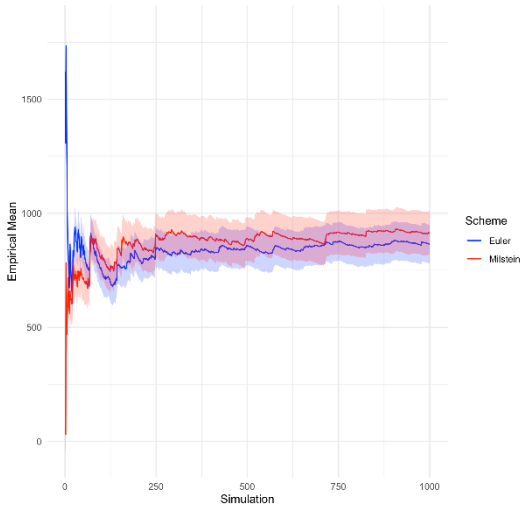


Figure 8: Evolution of Empirical Mean and Confidence Intervals

4.8. L^2 Error for Milstein scheme.

Similarly, I did the same for the L^2 error, as shown in Figure 9, while Table 2 shows the same result but in numerical form. The Milstein scheme has an error that converges faster compared to the Euler scheme. As N increases, leading to a decreasing Δt time step, the numerical approximation S_T^m becomes more accurate, leading to a smaller L^2 error. However, even though increasing N improves the accuracy of each simulation, the empirical mean is still subject to variability due to finite number of simulations M. Confidence intervals help to quantify this uncertainty.

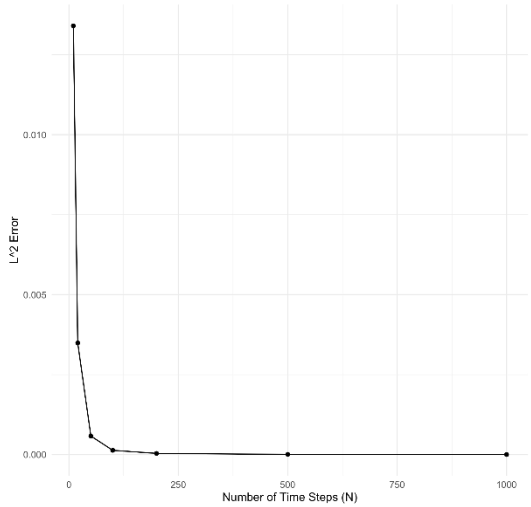


Figure 9: L2 error for Milstein Scheme as function of N

N	L2_Error
10	1.340390e-02
20	3.489025e-03
50	5.786265e-04
100	1.329387e-04
200	3.610572e-05
500	5.354944e-06
1000	1.431723e-06

Table 2: Variation of L2 error

5. Exercise 5: Topic 5: Monte Carlo integration.

5.1. True Value of Theta.

The first task of the exercise deals with computing the true value of theta. For this task, I used Wolfram Alpha, and I obtained $\theta \approx -0.132784$.

5.2. Determine $\tilde{\theta}$ with $Y \sim Exp(1/2)$.

To determine the $\tilde{\theta}$, I first defined the function as follows and then I simulated it with $n = 10^4$. The estimated value is $\theta \approx -0.1205654$.

```
# Monte Carlo integration using Y ~ Exp(1/2)
```

```r
monte_carlo_exp <- function(n) {
  Y <- rexp(n, rate = 1/2)
  g <- f(Y) / (1/2 * exp(-Y/2))
  mean(g)
}
```

5.3. Produce the relative plot.

Figure 10 shows the plot of the estimated $\tilde{\theta}$ with $Y \sim Exp(1/2)$ similarly to the one proposed in (11).

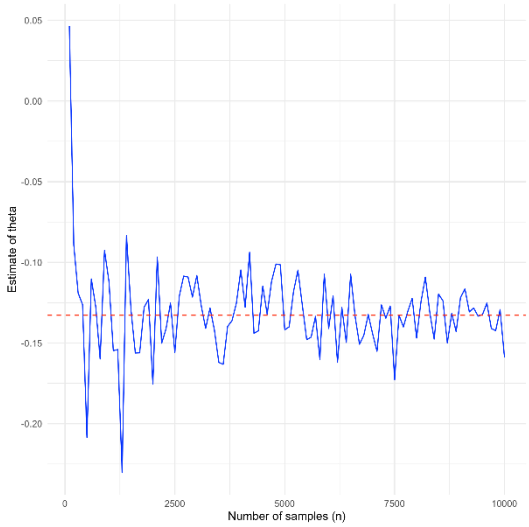


Figure 10: Monte Carlo Integration Estimates (Exp (1/2)) with true value in red

5.4. Determine $\tilde{\theta}$ with $Y \sim \chi^2_6$.

To determine the $\tilde{\theta}$, I first defined the function as follows, and then I simulated it with $n = 10^4$. The estimated value is $\theta \approx 0.1293532$.

```r
# Monte Carlo integration using Y ~ Chi-squared(6)
monte_carlo_chi <- function(n) {
  Y <- rchisq(n, df = 6)
  g <- f(Y) / (1/(2^(3) * gamma(3)) * Y^(3-1) * exp(-Y/2))
  mean(g)
}
```

5.5. Produce the relative plot.

Figure 11 shows the plot of the estimated $\tilde{\theta}$ with $Y \sim \chi^2_6$.

Figure 11: Monte Carlo Integration Estimates (Chi-squared (6)) with true value in red

5.6. Compare estimators for $n = 10^4$ and $M = 10^2$ realizations.

To complete the task, a new version of the estimator has been implemented as follows:

```
# Simulate M realizations of the estimators

simulate_realizations <- function(M, n, estimator) {
  realizations <- numeric(M)
  for (i in 1:M) {
    realizations[i] <- estimator(n)
  }
  return(realizations)
}
```

Moreover, the task requires comparing the two estimators based on distance to the real value and their variance. The Exp(1/2) has a mean of -0.131227 and a variance of 0.0001440485, while the Chi-squared has a mean of -0.1316119 and a variance of 0.0007547391 (cf., Figure 12).

From the result, I can conclude that the Exp(1/2) estimator has a slightly smaller bias, as its mean is closer to the true value compared to the Chi-squared estimator. Additionally, the Exp(1/2) estimator has a significantly lower variance than the Chi-squared estimator, which leads to less variability. Therefore, considering both the bias (distance to the true value) and the variance, the Exp(1/2) estimator is the better Monte Carlo estimator for this integral.

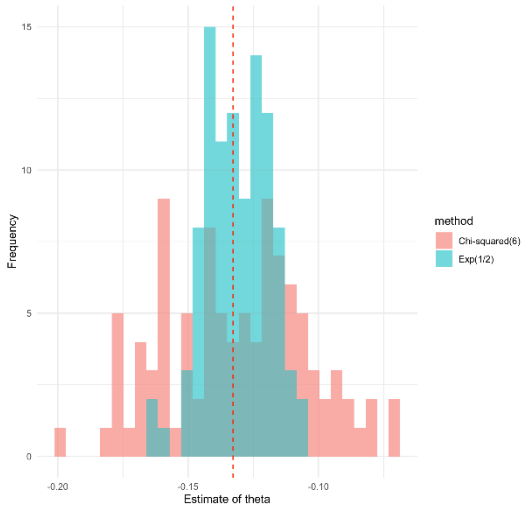


Figure 12: Distribution of Monte Carlo estimators

Slides Reference (ISO 690)

1. **Meddah, A.** *Stochastic Simulation. Topic 1: Simulation of “random” numbers from uniform distribution.* Linz : JKU, Week 2. p. Slide 29/41.

2. —. *Stochastic Simulation. Topic 1: Simulation of "random" numbers from uniform distribution.* Linz : JKU, Week 2. p. Slide 30/41.

3. —. *Stochastic Simulation. Topic 1: Simulation of "random" numbers from uniform distribution.* Linz : JKU, Week 2. p. Slide 31/40.

4. —. *Stochastic Simulation. Topic 5: Monte Carlo Methods.* Linz : JKU, Week 11. p. Slide 3/36.

5. —. *Stochastic Simulation. Topic 2: Simulation of “random” numbers from non-uniform distributions.* Linz : JKU, Week 3. p. Slide 19/31.

6. —. *Stochastic Simulation. Topic 2: Simulation of "random" numbers from non-uniform distribution.* Linz : JKU, Week 3. p. Slide 21/31.

7. —. *Stochastic Simulation. Topic 2: Simulation of "random" numbers from non-uniform distributions.* Linz : JKU, Week 4. p. Slide 13/23.

8. —. *Stochastic Simulation. Topic 3: Simulation of stochastic processes. Random walk and Markov chains.* Linz : JKU, Week 5. p. Slide 14/33.

9. —. *Stochastic Simulation. Topic 3: Simulation of stochastic processes. Random walk and Markov chains.* Linz : JKU, Week 5. p. Slide 32/33.

10. —. *Stochastic Simulation. Topic 4: Simulation of stochastic differential equations (SDEs).* Linz : JKU, Week 10. p. Slide 21/25.

11. —. *Stochastic Simulation. Topic 5: Monte Carlo Methods.* Linz : JKU, Week 11. p. Slide 13/36.

List of Figures