**National University of the Altiplano**
**Faculty of Statistical Engineering and Computer Science**
**Professor:** Fred Torres Cruz
**Author:** Flores Turpo Jorge Luis
**Repository:** `https://github.com/Yorchisflrs/libreria_r`

# Assignment - No. 05: Implementation of the Mersenne Twister Generator in R

## 1. Introduction

The Mersenne Twister generator is one of the most widely used and reliable pseudorandom number generators today. It was developed by Makoto Matsumoto and Takuji Nishimura in 1997 and is named after its period, which is a Mersenne prime ($2^{19937} - 1$).

## 2. Theoretical Background

The Mersenne Twister algorithm was developed to overcome the limitations of previous generators, offering an extremely long period and good statistical properties. Its name comes from Mersenne prime numbers, as the algorithm's period is $2^{19937} - 1$. The algorithm uses a state matrix of 624 integers and a series of bitwise operations to transform the state and produce sequences of pseudorandom numbers with high equidistribution.

The main process consists of two phases:

- **Initialization**: The internal state is set from a seed.
- **Generation**: The state is updated and a series of transformations are applied to obtain the pseudorandom numbers.

## 3. Implementation Description

The implementation in R uses the `bitops` package for the bitwise operations required by the algorithm. The code is structured into three main functions:

### 3.1. Initialization (mt_init)

This function initializes the internal state of the generator:

```r
mt_init <- function(seed = 5489) {
  .mt_env$mt <- numeric(624)
  .mt_env$idx <- 1L
  .mt_env$mt[1] <- as.numeric(bitwAnd(as.integer(seed), 0xFFFFFFFF))

  for (i in 2:624) {
    temp <- bitwXor(.mt_env$mt[i-1], bitwShiftR(.mt_env$mt[i-1], 30))
    .mt_env$mt[i] <- as.numeric(bitwAnd(1812433253 * temp + (i-1), 0xFFFFFFFF))
  }
}
```

## 3.2. Number Generation (mt_generate)

The main function that generates the pseudorandom numbers:

```r
mt_generate <- function(n = 1) {
  # The full code includes the state transformation
  # and the generation of numbers in the range [0,1)
}
```

## 4. Full Implementation Code

The main code used in R is shown below:

```r
if (!require("bitops", quietly = TRUE)) {
  install.packages("bitops")
  library(bitops)
}

.mt_env <- new.env()

mt_init <- function(seed = 5489) {
  .mt_env$mt <- numeric(624)
  .mt_env$idx <- 1L
  .mt_env$mt[1] <- as.numeric(bitwAnd(as.integer(seed), 0xFFFFFFFF))
  for (i in 2:624) {
    temp <- bitwXor(.mt_env$mt[i-1], bitwShiftR(.mt_env$mt[i-1], 30))
    .mt_env$mt[i] <- as.numeric(bitwAnd(1812433253 * temp + (i-1), 0xFFFFFFFF))
  }
}

mt_generate <- function(n = 1) {
  if (.mt_env$idx > 624) {
    for (i in 1:624) {
      x <- bitwOr(
        bitwAnd(as.integer(.mt_env$mt[i]), 0x80000000),
        bitwAnd(as.integer(.mt_env$mt[i %% 624 + 1]), 0x7FFFFFFF)
      )
      xA <- bitwShiftR(x, 1)
      if (bitwAnd(x, 1)) xA <- bitwXor(xA, 0x9908B0DF)
      .mt_env$mt[i] <<- bitwXor(as.integer(.mt_env$mt[(i + 396) %% 624 + 1]), xA)
    }
    .mt_env$idx <<- 1
  }
  res <- numeric(n)
  for (i in 1:n) {
    y <- as.integer(.mt_env$mt[.mt_env$idx])
    y <- bitwXor(y, bitwShiftR(y, 11))
    y <- bitwXor(y, bitwAnd(bitwShiftL(y, 7), 0x9D2C5680))
    y <- bitwXor(y, bitwAnd(bitwShiftL(y, 15), 0xEFC60000))
    y <- bitwXor(y, bitwShiftR(y, 18))
    res[i] <- min(y / 4294967296.0, 0.999999999)
    .mt_env$idx <<- .mt_env$idx + 1
  }
  res
}

mt_set_seed <- function(seed) {
  if (!is.numeric(seed)) stop("Seed must be numeric")
  mt_init(seed)
}
```

## 5. Usage Example

To use the generator:

```
# Initialize with a seed
mt_set_seed(123)

# Generate 5 random numbers
results <- mt_generate(5)
```

## 6. Advantages and Applications

- Very long period
- Good statistical properties
- Efficient in terms of memory and speed
- Ideal for Monte Carlo simulations
- Useful in cryptography and statistical modeling

## 7. Conclusions and Possible Improvements

The implementation of the Mersenne Twister in R is efficient and meets international standards for pseudorandom number generation. For critical applications, it is recommended to compare the results with other generators and perform additional statistical tests (e.g., randomness tests). As an improvement, a formal R package could be created and more analysis and visualization functions could be added.

## 8. References

1. Matsumoto, M., & Nishimura, T. (1998). Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. ACM Transactions on Modeling and Computer Simulation, 8(1), 3-30.

2. R Core Team. (2023). R: A Language and Environment for Statistical Computing.