Homework 2 - STAT 5362 Statistical Computing

Sen Yang[*]

14 September 2018

Abstract

This is **Homework 2** for STAT 5362 Statistical Computing. HW2 has 2 questions. The first question is Exercise 2 of Chapter 1, and the second question is Exercise 3 of Chapter 1. For the first question, I use nested *for* loops to get the approximation value by the Monte Carlo methods, then repeat the experiment for 100 times and draw corresponding boxplot. Fot the second question, by using the 64-bit double precision floating point arithmetic, I show the listed 4 numbers and check them by R.

Keywords: Template; R Markdown; **bookdown**; **knitr**; **Pandoc**

1 Chapter 1 - Exercise 2

1.1 Math Equations

The distribution function of $N(0, 1)$ is,

$$\Phi(t) = \int_{-\infty}^{t} \frac{1}{\sqrt{2\pi}} e^{-y^2/2} dy, \tag{1}$$

Consider approximation of the distribution by the Monte Carlo methods:

$$\hat{\Phi}(t) = \frac{1}{n} \sum_{i=1}^{n} I(X_i \leq t), \tag{2}$$

1.2 Table

Experiment with the approximation at $n \in \{10^2, 10^3, 10^4\}$ at $t \in \{0.0, 0.67, 0.84, 1.28, 1.65, 2.32, 2.58, 3.09, 3.72\}$ to form a table. I apply `set.seeds = (100)` before getting random standard normal distribution. Table 1 shows theoretical value and all approximations with different n with Seed 100.

1.3 Figures

Repeat the experiment 100 times. Draw box plots of the 100 approximation errors at each t for each n. The result is shown in Figure 1.

It can be concluded that the approximation is more precise and the variation of errors is decreasing with n increasing. Meanwhile, the error goes around 0 in general.

[*] sen.2.yang@uconn.edu; M.S. student at Department of Statistics, University of Connecticut.

Table 1: Table of Theoretical values and Approximations with Seed 100

	Theoretical Value	n = 100	n = 1000	n = 10000
t = 0.0	0.5000000	0.52	0.483	0.5033
t = 0.67	0.7485711	0.75	0.736	0.7504
t = 0.84	0.7995458	0.81	0.794	0.8030
t = 1.28	0.8997274	0.87	0.897	0.9044
t = 1.65	0.9505285	0.93	0.950	0.9519
t = 2.32	0.9898296	0.98	0.986	0.9900
t = 2.58	0.9950600	0.99	0.994	0.9950
t = 3.09	0.9989992	1.00	0.998	0.9994
t = 3.72	0.9999004	1.00	1.000	0.9998

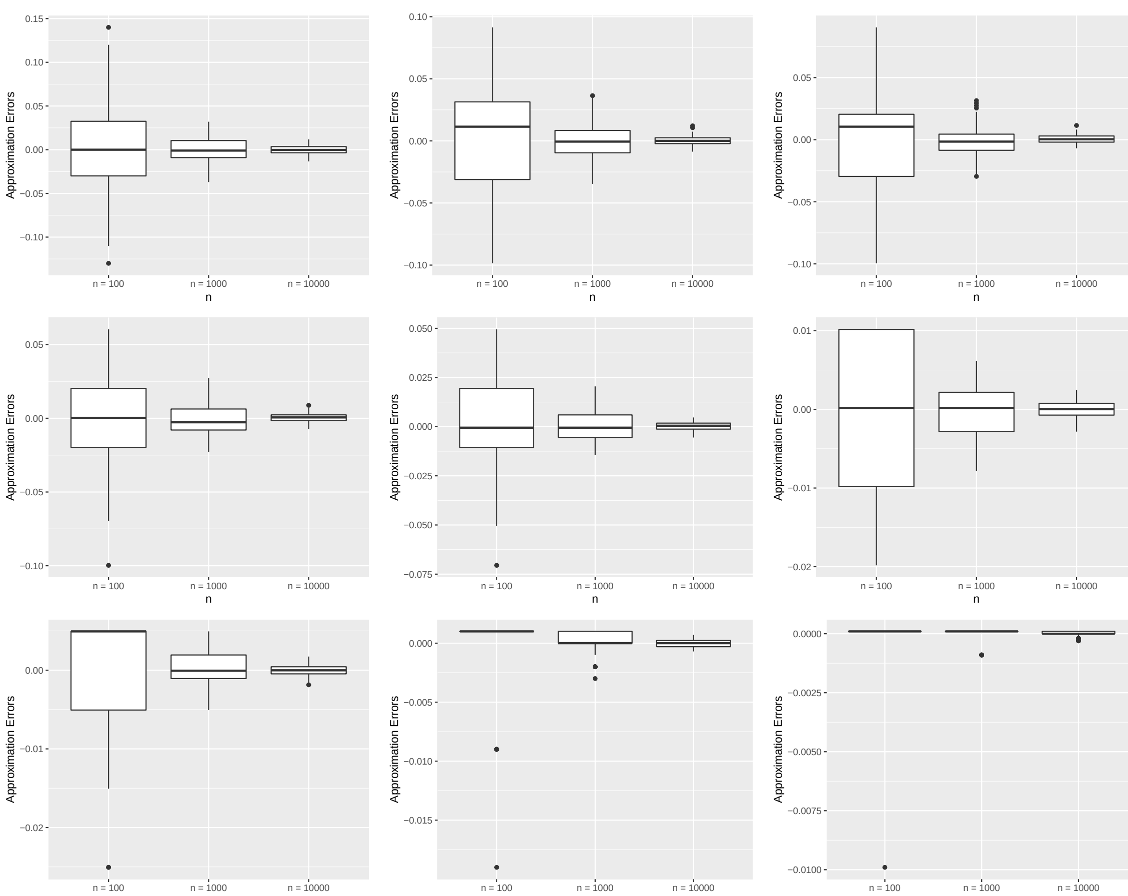

Figure 1: Boxplot of Approximation Errors for Each t

1.4 Code Chunk

1.4.1 Code for Tables

```
t <- c(0.0, 0.67, 0.84, 1.28, 1.65, 2.32, 2.58, 3.09, 3.72)
n <- c(10^2, 10^3, 10^4)
n_le <- 0
# Repeat the experiment 100 times and record the biases
table_bias100 <- matrix(0,300,9)
# Build the probability matrices and bias matrices by the Monte Carlo methods
for (m in 1:100) {
  set.seed(m)
  table_p <- matrix(0,4,length(t))
  table_p[1,] <- t
  table_bias <- table_p
  for (i in 1:length(n)) {
    a <- rnorm(n[i])
      for (j in 1:length(t)) {
        for (k in 1:n[i]) {
          if (a[k] <= t[j]) {
            n_le <- n_le + 1
          }
        }
        table_p[1,j] <- pnorm(t[j])
        table_p[1+i,j] <- n_le/n[i]
        n_le <- 0
        table_bias[1,j] <- pnorm(t[j])
        table_bias[1+i,j] <- table_p[1+i,j]-table_p[1,j]
        table_bias100[100*(i-1)+m,j] <- table_bias[1+i,j]
      }
  }
}
# print a table of probability with seed(100)
df_p <- as.data.frame(table_p)
colnames(df_p) <- c("t = 0.0", "t = 0.67", "t = 0.84", "t = 1.28", "t = 1.65", "t = 2.32", "t =
rownames(df_p) <- c("Theoretical Value", "n = 100", "n = 1000", "n = 10000")
df_p <- t(df_p)
knitr::kable(df_p, caption = 'Table of Theoretical values and Approximations with Seed100', boo
```

1.4.2 Code for boxplots

```
# Boxplots of n = 100, 1000, 10000 for each t
library(ggplot2)
x <- as.data.frame(rep(c("n = 100", "n = 1000", "n = 10000"), each = 100))
for (j in 1:length(t)) {
```

```
  table_boxplot <- cbind(table_bias100[1:300,j],x)
  df_boxplot <- as.data.frame(table_boxplot)
  colnames(df_boxplot) <- c("e", "n")
  print(ggplot(df_boxplot, aes(n, e)) + ylab("Approximation Errors") + geom_boxplot())
}
```

2 Chapter 1 - Exercise 3

The real value assumed by a given 64-bit double-precision datum is:

$$(-1)^{sign}(1+\sum_{i=1}^{52} b_{52-i}2^{-i}) \times 2^{exponent-1023} \tag{3}$$

2.1 .Machine$double.xmax

`.Machine$double.xmax` is the largest normalized floating-point number.

By binary digits, it can be shown as:

$$0\ \ 11111111110\ \ 1111111111111111111111111111111111111111111111111111_2$$

By decimal digits, it can be shown as:

$$(-1)^0(1+(1-2^{-52}))\times 2^{1023}$$

Calculated by R, the answer is

```
## [1] 1.797693e+308
```

2.2 .Machine$double.xmin

`.Machine$double.xmin` is the smallest non-zero normalized floating-point number.

By binary digits, it can be shown as:

$$0\ \ 00000000001\ \ 0000000000000000000000000000000000000000000000000000_2$$

By decimal digits, it can be shown as:

$$(-1)^0(1+1)\times 2^{-1023}$$

Calculated by R, the answer is

```
## [1] 2.225074e-308
```

2.3 .Machine$double.eps

`.Machine$double.eps` is the smallest positive floating-point number x such that $1 + x \neq 1$. By decimal digits, it can be shown as:

$$2^{-52}$$

Calculated by R, the answer is

```
## [1] 2.220446e-16
```

2.4 .Machine$double.neg.eps

`.Machine$double.neg.eps` is a small positive floating-point number x such that $1 - x \neq 1$. By decimal digits, it can be shown as:

$$2^{-53}$$

Calculated by R, the answer is

```
## [1] 1.110223e-16
```