HW2 - Exercise2 and Exercise3

JooChul Lee

13 September 2018

Abstract

This is the second homework of STAT-5361. There are two parts; One is the Exercise 2 and the other is the Exerxise 3. For the exercise2, I consider approximation of the standard normal distribution by the Monte Carlo method and compare the true value. Also, I repeat the experiment 100 times and draw box plots for the bias for all t. For the exercise3, I define .Machine$double.xmax, .Machine$double.xmin, .Machine$double.eps, and .Machine@double.neg.eps by using the 64-bit double precision floating point arithmetic.

Contents

1 Exercise 2

1.1 Math Equations

We consider approximation of the distribtuion function of $N(0,1)$,

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

by the Monte Carlo methods which is

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

where X_i's are iid $N(0,1)$ variables.

1.2 Tables and Figures

Table 1 is the results for Experiment with the approximation at $n \in \{102, 103, 104\}$ at $t \in \{0.0, 0.67, 0.84, 1.28, 1.65, 2.32, 2.58, 3.09, 3.72\}$. Firgure 1, 2, and 3 is the box plots of bias for all t after repeating the experiment 100 times.

Table 1: Table for the experiment with the approximation

	t = 0.0	t = 0.67	t = 0.84	t = 1.28	t = 1.65	t = 2.32	t = 2.58	t = 3.09	t = 3.72
n = 100	0.5400	0.7900	0.8100	0.9300	0.9500	0.9900	1.0000	1.0000	1.0000
n = 1000	0.4950	0.7550	0.8040	0.8990	0.9450	0.9900	0.9970	0.9990	1.0000
n = 10000	0.4951	0.7425	0.7956	0.8975	0.9495	0.9908	0.9955	0.9988	0.9999
True Value	0.5000	0.7486	0.7995	0.8997	0.9505	0.9898	0.9951	0.9990	0.9999

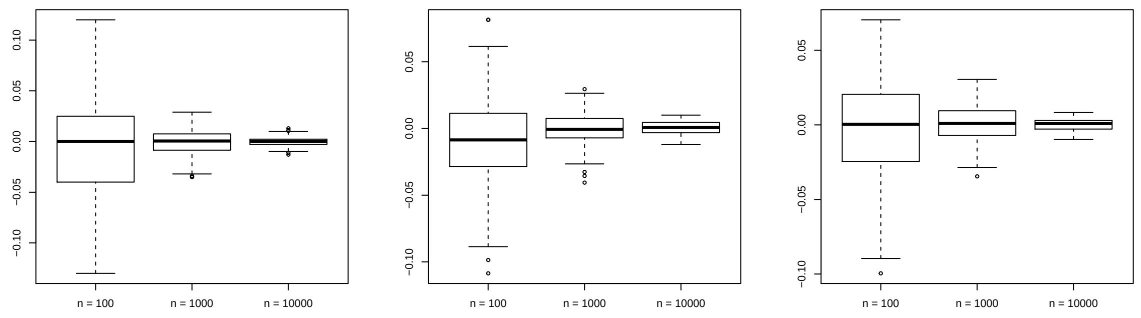

Figure 1: Left : t = 0.0, Middle : t = 0.67, Right : t = 0.84

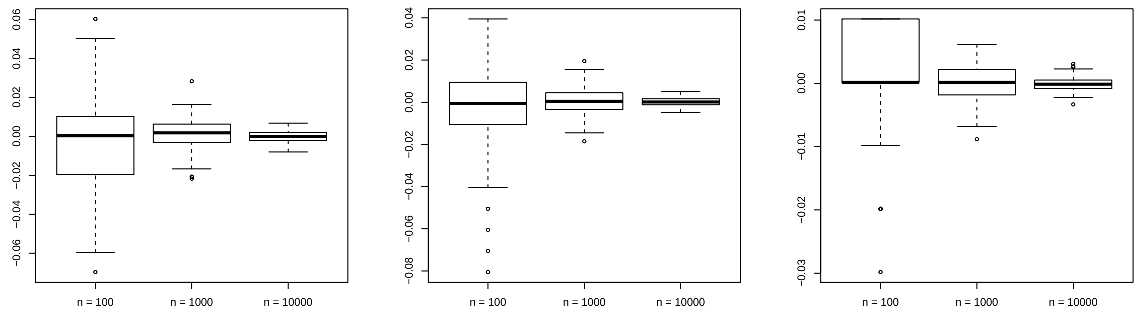

Figure 2: Left : t = 1.28, Middle : t = 1.65, Right : t = 2.32

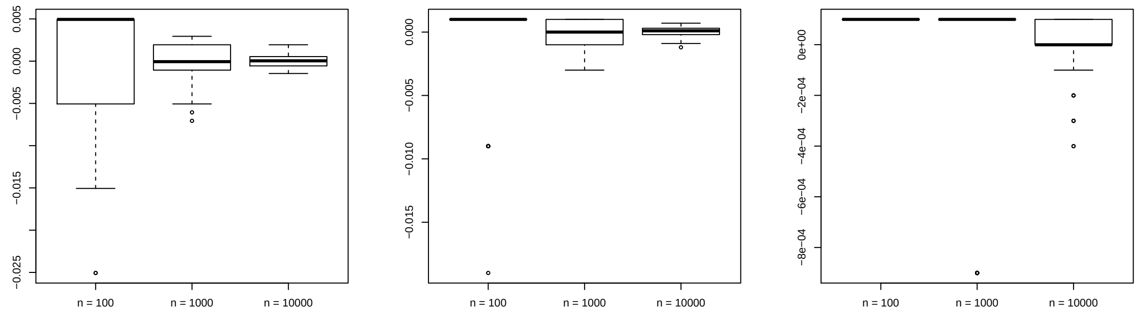

Figure 3: Left : t = 2.58, Middle : t = 3.09, Right : t = 3.72

1.3 Result and Summary

In the table 1, when the size of sample increases, the approximated vaules are closer to the true values. From the figures which are the boxplots for bias, we can see that when size of sample inceases, the variation of the bias is usually decreasing and the median of bias is closer to zero.

1.4 Code Chunk

This is the code for table 1.

```r
set.seed(5361)
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)
approx = matrix(0,3,length(t))
for(i in 1:3)
{
   dat = rnorm(n[i])
   for(j in 1:length(t))
      approx[i,j] = mean( dat <= t[j])
}
result = as.data.frame(rbind(approx,pnorm(t)))
row.names(result) = c('n = 100','n = 1000','n = 10000','True Value')
```

This is the core for figures 1, 2, and 3.

```r
re = 100
approx = matrix(0,re,length(t))
RepDat = vector("list", length(n))
for(i in 1:3)
{
   for(j in 1:re)
   {
      dat = rnorm(n[i])
      for(k in 1:length(t))
         approx[j,k] = mean( dat <= t[k]) - pnorm(t[k])
   }
   RepDat[[i]] = approx
}

#par(mfrow = c(3,3))
#for(i in 1:9)
#   boxplot(RepDat[[1]][,i],RepDat[[2]][,i],RepDat[[3]][,i],
#           names=c('n = 100','n = 1000','n = 10000'))
```

2 Exercise 3

Explain how `.Machine$double.xmax`, `.Machine$double.xmin`, `.Machine$double.eps`, and `.Machine$double.neg.eps` are defined using the 64-bit double precision floating point arithmetic.

1.1 `.Machine$double.xmax` : 64-bit double precision floating point arithmetic

$$0\ 11111111110\ 1111111111111111111111111111111111111111111111111111_2$$

In the part of the exponent, there are all 1 for the 11bit, it is used to represent infinity. Because `.Machine$double.xmax` is the largest normalized floating-point number, it should be $2^{1023}(1+(1-2^{-52}))$ which can be expressed like above using 64-bit double precision floating point arithmetic.

```r
.Machine$double.xmax
```

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

1.2 `.Machine$double.xmin` : 64-bit double precision floating point arithmetic

$$0\ 00000000001\ 0000000000000000000000000000000000000000000000000000_2$$

In the part of the exponent and significand, there are all 0 for the 11 bit and 52 bit , it is used to represent zero. Because `.Machine$double.xmin` is the smallest non-zero normalized floating-point number, it should be $2^{-1022} * 1$ which can be expressed like above using 64-bit double precision floating point arithmetic.

```r
.Machine$double.xmin
```

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

1.3 `.Machine$double.eps` : 64-bit double precision floating point arithmetic

$$0\ 01111001011\ 0000000000000000000000000000000000000000000000000000_2$$

`.Machine$double.eps` is the smallest positive floating-point number x such that 1 + x!= 1. It means that by 64-bit double precision floating point, it should be $2^{-52} * 1$ which can be expressed like above using 64-bit double precision floating point arithmetic.

```r
.Machine$double.eps
```

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

1.4 `.Machine$double.neg.eps` : 64-bit double precision floating point arithmetic

$$0\ 01111001010\ 0000000000000000000000000000000000000000000000000000_2$$

`.Machine$double.neg.eps` is a small positive floating-point number x such that 1 - x!= 1. It means that by 64-bit double precision floating point, it should be $2^{-53} * 1$ which can be expressed like above using 64-bit double precision floating point arithmetic.

```
.Machine$double.neg.eps
```

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