HW2

Cheng Huang[*]

14 September 2018

Abstract

This Model is designed to evaluate the Monte Carlo method when approximate the distribution function of $N(0,1)$. The sample size are $n \in \{10^2, 10^3, 10^4\}$, and the CDF is evaluated at $t \in \{0.0, 0.67, 0.84, 1.28, 1.65, 2.32, 2.58, 3.09, 3.72\}$. A table makes comparison between the true value and the approxiamte value of the CDF for all sample size. The box plots demonstrate how the bias distribution change along different t.

Problem 2

Methodology

The Monte Carlo methods is using:

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

to estimate the distribution function of $N(0,1)$.
This calculation is achieved by define a function to calculate the empirical cdf value:

```
tlist <- c(0.0, 0.67, 0.84, 1.28, 1.65, 2.32, 2.58, 3.09, 3.72)
cdf.cal.fun <- function(n, tlist){
  x <- rnorm(n, mean = 0, sd = 1)
  cdf <- double(length(tlist))
  for(t in tlist){
    cdf[which(tlist == t)] <- sum(x <= t) /n
  }
  cdf
}
```

Result and Conclusions

Table

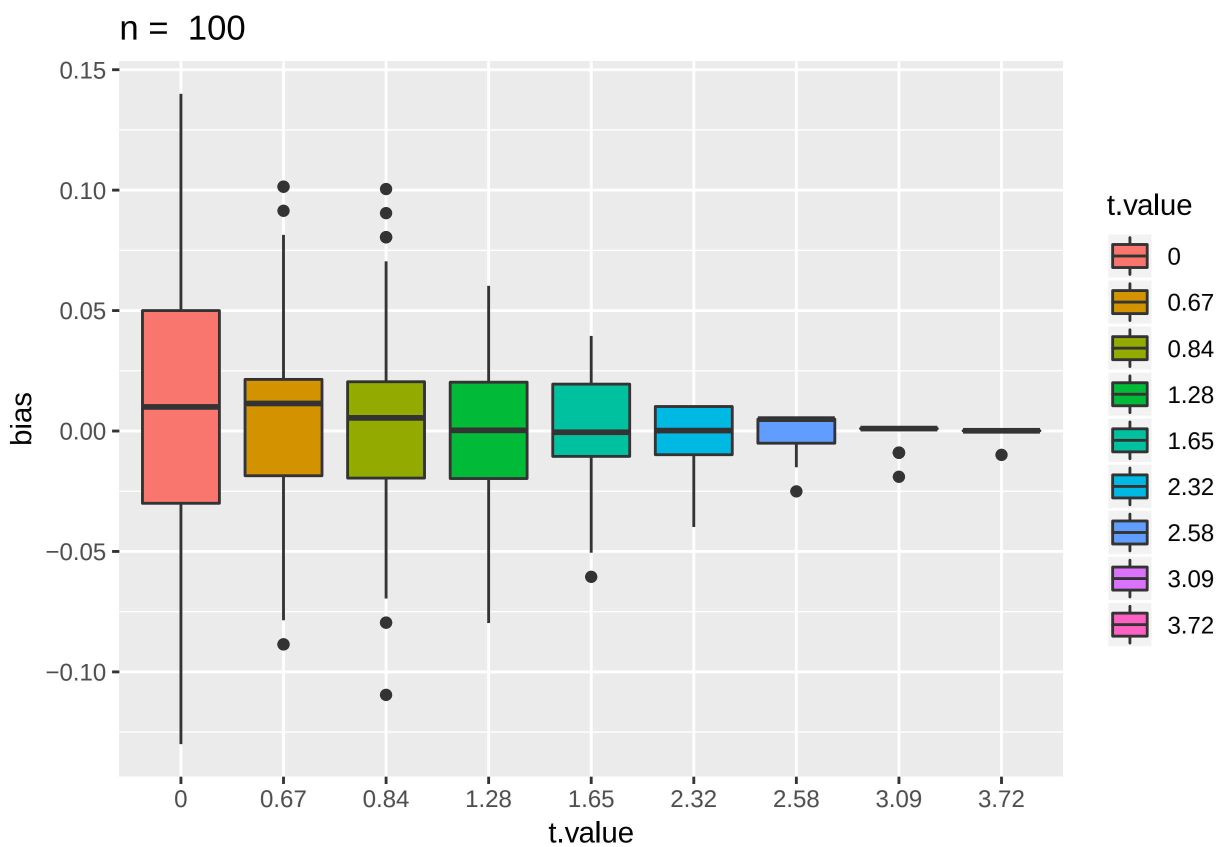
The first result is to show the table comparison between the true value of CDF and the approximate one.

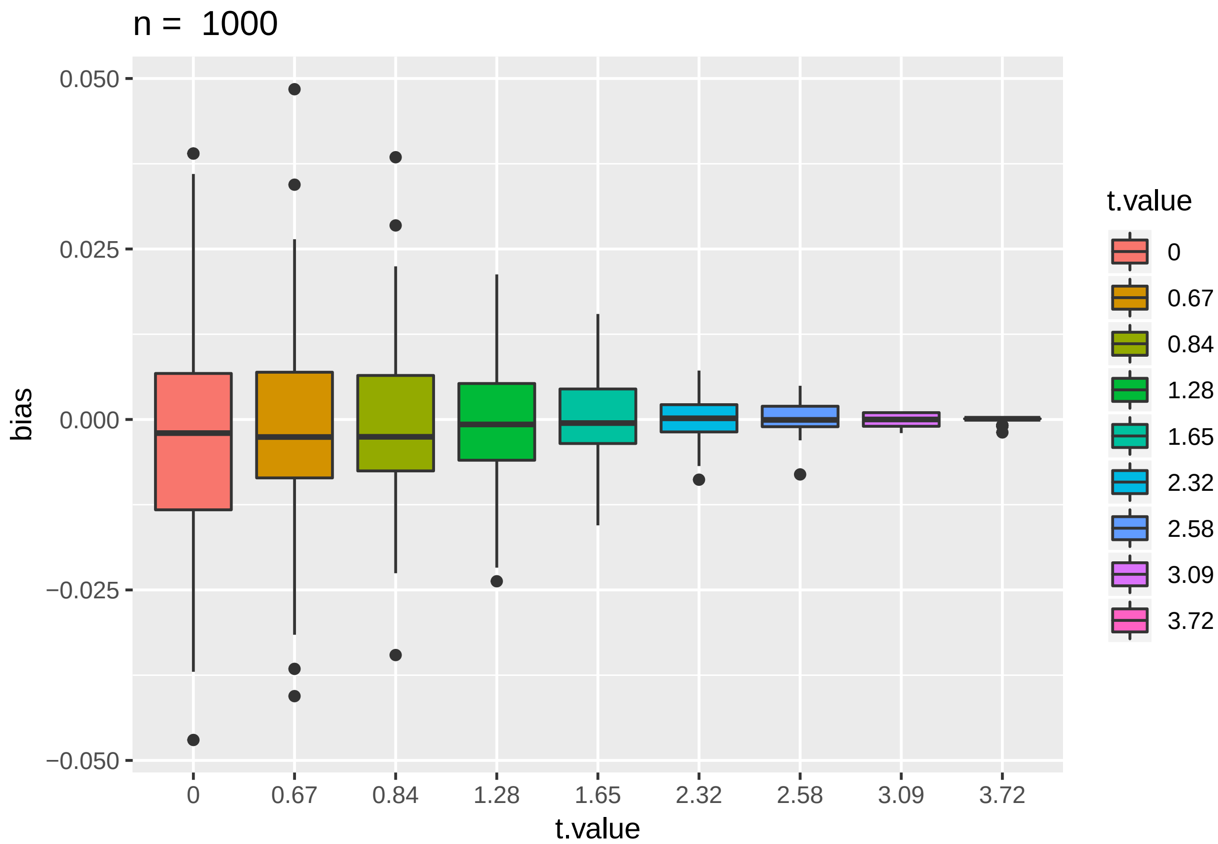
[*]; Ph.D. student at Department of Statistics, University of Connecticut.

```
##    t value true value of cdf n = 100 n = 1000 n = 10000
## 1:    0.00          0.5000000    0.48    0.492    0.5054
## 2:    0.67          0.7485711    0.74    0.752    0.7466
## 3:    0.84          0.7995458    0.79    0.798    0.7979
## 4:    1.28          0.8997274    0.90    0.900    0.9031
## 5:    1.65          0.9505285    0.95    0.939    0.9519
## 6:    2.32          0.9898296    1.00    0.983    0.9891
## 7:    2.58          0.9950600    1.00    0.995    0.9948
## 8:    3.09          0.9989992    1.00    0.999    0.9989
## 9:    3.72          0.9999004    1.00    1.000    0.9999
```

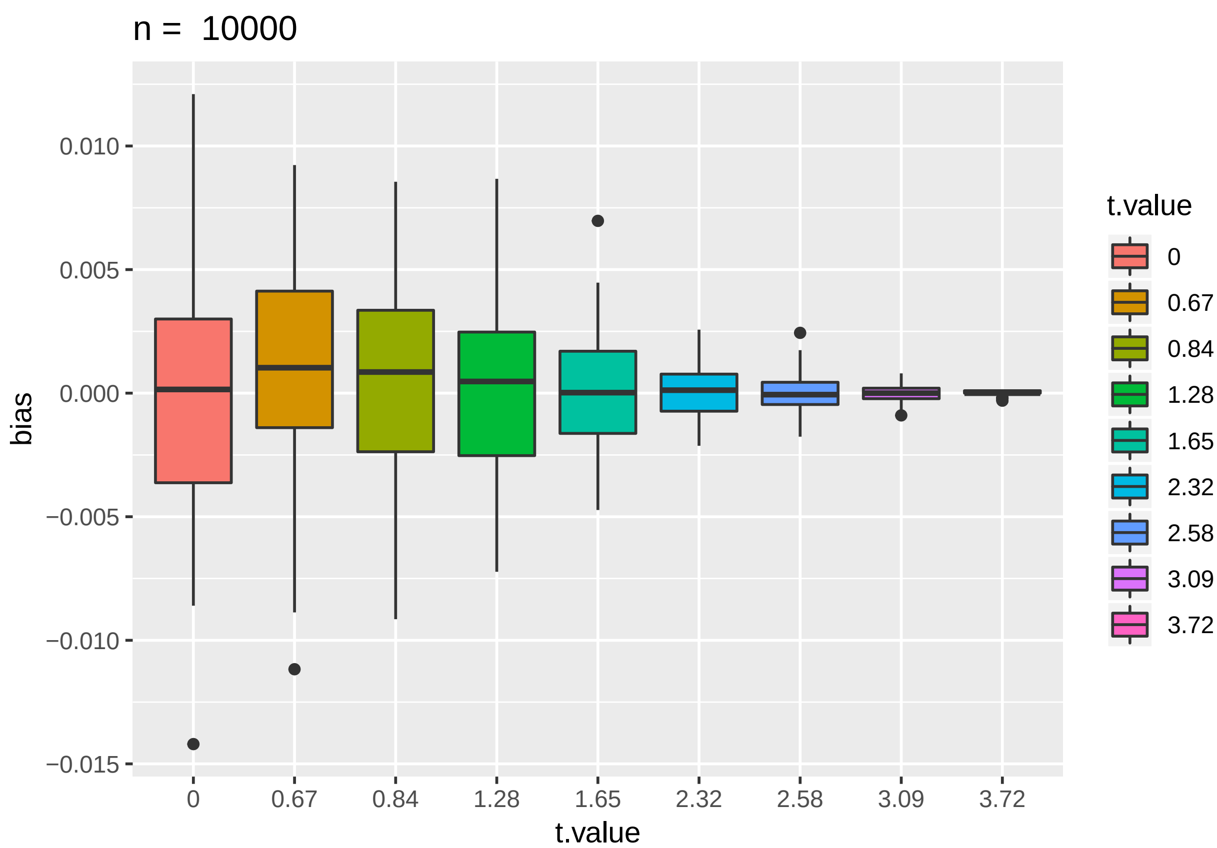
Bias graph

The following box-plots gives the bias under different sample size n.
As expected, few numbers generated when t is large, so the distribution of the bias becomes less spread for large t.





n = 1000
bias
0.050
0.025
0.000
−0.025
−0.050
0
0.67
0.84
1.28
1.65
2.32
2.58
3.09
3.72
t.value
t.value
0
0.67
0.84
1.28
1.65
2.32
2.58
3.09
3.72



Problem 3

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?
The reprensentation of the 64-bit double-precision number is given by

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

- sign bit: 1 bit
- Exponent: 11 bits
- Significand : 53 bits.

The exponent takes 11 bits, there are total $2^{11} = 2048$ possible values represented, it is centored as e - 1023 is representing integer from -1024 to 1023.

`.Machine$double.xmax`
the largest normalized floating-point number. Typically, it is equal to (1 - double.neg.eps) * double.base ^ double.max.exp, but on some machines it is only the second or third largest

such number, being too small by 1 or 2 units in the last digit of the significand. Normally 1.797693e+308. Note that larger unnormalized numbers can occur.

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

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

```
(1 + sum(2 ^ -c(1:52))) * 2 ^ 1023
```

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

`.Machine$double.xmin`
the smallest non-zero normalized floating-point number, a power of the radix, i.e., double.base ^ double.min.exp. Normally 2.225074e-308.

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

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

```
(1 - 2 ^ -53) * 2 ^ -1022 == .Machine$double.xmin
```

```
## [1] TRUE
```

```
(1 - .Machine$double.neg.eps) * .Machine$double.base ^ .Machine$double.min.exp == .Machine$doul
```

```
## [1] TRUE
```

`.Machine$double.eps`
the smallest positive floating-point number x such that 1 + x != 1. It equals double.base ^ ulp.digits if either double.base is 2 or double.rounding is 0; otherwise, it is (double.base ^ double.ulp.digits) / 2. Normally 2.220446e-16.

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

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

```
.Machine$double.base ^ .Machine$double.ulp.digits == .Machine$double.eps
```

```
## [1] TRUE
```

```r
2 ^ -52 == .Machine$double.eps
```

```
## [1] TRUE
```

`.Machine@double.neg.eps`
a small positive floating-point number x such that 1 - x != 1. It equals double.base ^ double.neg.ulp.digits if double.base is 2 or double.rounding is 0; otherwise, it is (double.base ^ double.neg.ulp.digits) / 2. Normally 1.110223e-16. As double.neg.ulp.digits is bounded below by -(double.digits + 3), double.neg.eps may not be the smallest number that can alter 1 by subtraction.

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

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

```r
.Machine$double.base ^ .Machine$double.neg.ulp.digits == .Machine$double.neg.eps
```

```
## [1] TRUE
```

```r
2 ^ -53 == .Machine$double.neg.eps
```

```
## [1] TRUE
```

Reference

[Double-precision floating-point format]https://en.wikipedia.org/wiki/Double-precision_floating-point_format
[jun-yan/stat-5361]https://github.com/jun-yan/stat-5361
[ggplot2 boxplot]
http://www.sthda.com/english/wiki/ggplot2-box-plot-quick-start-guide-r-software-and-data-visualization