Monte Carlo Methods and Computer Arithmetics

HW 2 of STAT 5361 Statistical Computing

*Biju Wang**

09/10/2018

Abstract

Use Monte Carlo Methods to approch $\Phi(t)$ and explain some computer arithmetics.

1 Monte Carlo Methods

1.1 Principles

The CDF of standard norm distribution is

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

by the Monte Carlo methods

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

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

1.2 Approximation Outcomes

The approximation is implemented at $n \in \{10^2, 10^3, 10^4\}$ at $t \in \{0.0, 0.67, 0.84, 1.28, 1.65, 2.32, 2.58, 3.09, 3.72\}$. The outcome table with true values is shown below.

Table 1: Approximation Outcomes with True Values

$\hat{\Phi}(t)$	$t = 0.0$	$t = 0.67$	$t = 0.84$	$t = 1.28$	$t = 1.65$	$t = 2.32$	$t = 2.58$	$t = 3.09$	$t = 3.72$
$n = 10^2$	0.4600	0.7400	0.8200	0.9100	0.9700	0.9900	1.0000	1.0000	1.0000
$n = 10^3$	0.5210	0.7390	0.7940	0.8920	0.9360	0.9900	0.9960	0.9990	0.9990
$n = 10^4$	0.5056	0.7496	0.7937	0.8967	0.9521	0.9910	0.9960	0.9994	1.0000
$\Phi(t)$	0.5000	0.7486	0.7995	0.8997	0.9505	0.9898	0.9951	0.9990	0.9999

```r
n <- c(10^2, 10^3, 10^4)
t <- c(0.0, 0.67, 0.84, 1.28, 1.65, 2.32, 2.58, 3.09, 3.72)
truth <- pnorm(t)

table <- matrix(NA, 3, 9)

set.seed(1)
sample <- list()
for (i in 1:length(n)) {
  sample[[i]] <- rnorm(n[i])
```

*bijuwang@uconn.edu

```
}

for (i in 1:length(n)) {
  for (j in 1:length(t)) {
    table[i, j] <- sum(sample[[i]] <= t[j])/n[i]
  }
}
```

1.3 Box Plots of the Bias

For each n and t, we have $\hat{\Phi}(t)$ and the true value $\Phi(t)$, then we can calculate the bias. We repeat the experiments 100 times and will obtain 100 biases. Thus box plots can be drawn.

```
## Using  as id variables
## Using  as id variables
## Using  as id variables
```

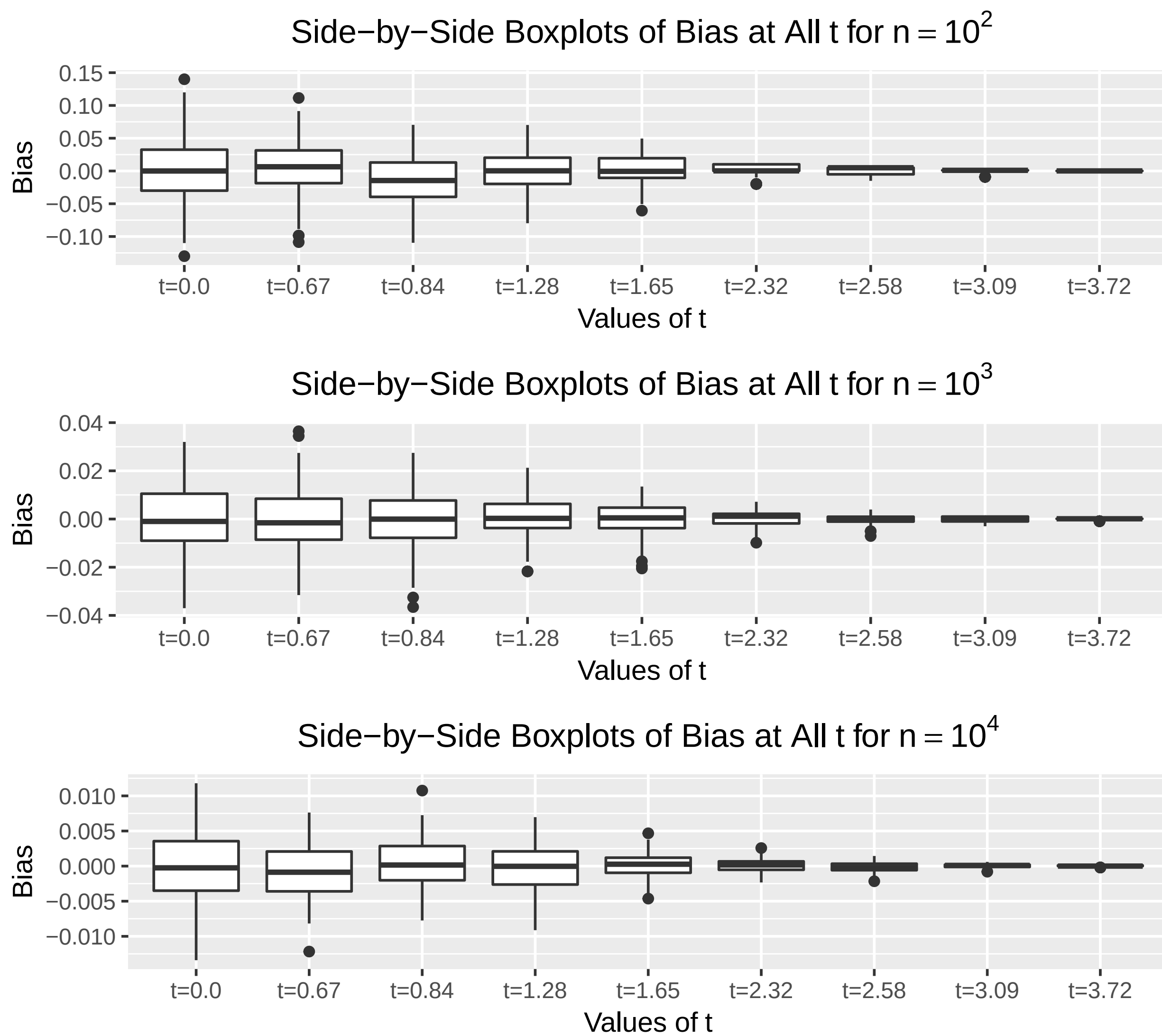

Figure 1: Side-by-Side Boxplots of Bias at all t for Different n

2 Computer Arithmetics

2.1 .Machine$double.xmax

The value of .Machine$double.xmax is

```
options(digits = 20)
.Machine$double.xmax
```

```
## [1] 1.7976931348623157081e+308
```

It can also be calculated by

```
u <- 0
for (i in 1L:53) u <- u + 2^(-i)
u * 2 * 2 ^ 1023
```

```
## [1] 1.7976931348623157081e+308
```

Thus, it is defined by setting sign bit to 1, significand to $\sum_{i=1}^{53} 2^{-i}$, exponent to 1024.

2.2 .Machine$double.xmin

The value of .Machine$double.xmin is

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

```
## [1] 2.2250738585072013831e-308
```

It can also be calculated by

```
2 ^ (-1022)
```

```
## [1] 2.2250738585072013831e-308
```

Thus, it is the smallest non-zero normalized floating-point number, a power of the radix.

2.3 .Machine$double.eps

The value of .Machine$double.eps is

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

```
## [1] 2.2204460492503130808e-16
```

It can also be calculated by

```
2 ^ (-52)
```

```
## [1] 2.2204460492503130808e-16
```

Thus, it is the smallest positive floating-point number x such that $1 + x\ != 1$.

2.4 .Machine$double.neg.eps

The value of .Machine$double.neg.eps is

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

```
## [1] 1.1102230246251565404e-16
```

It can also be calculated by

```
2 ^ (-53)
```

```
## [1] 1.1102230246251565404e-16
```

Thus, it is the smallest positive floating-point number x such that $1 - x \, ! = 1$.