STAT680_HW1

Yudi Zhang, Wangqian Ju

9/24/2021

Q1

a)

We have $g(X) = X^{\alpha-1}$ and $X \sim \text{Exp}(1)$

```
n <- 10000
u <- runif(n)

# obtain the Exp(1) random variable
ee <- -log(1-u)

alpha <- 1.89

e.gamma <- ee^(alpha - 1) %>% mean

# check the result using gamma function
c(e.gamma, gamma(alpha))
```

```
## [1] 0.9513581 0.9583793
```

b)

$$
\begin{aligned}
Var(\frac{1}{n}\sum X_i^{\alpha-1}) &= \frac{1}{n}Var(X^{\alpha-1}) \\
&= \frac{1}{n}(E[X^{2(\alpha-1)}] - [E(X^{\alpha-1})]^2) \\
&= \frac{1}{n}\left(\Gamma(2\alpha-1) - [\Gamma(\alpha)]^2\right)
\end{aligned}
$$

c)

```
estimate_gamma <- function(alpha, n = 1000) {
  u <- runif(n)
  ee <- -log(1-u)

  tt <- ee^(alpha - 1)
```

```
  est <- mean(tt)
  sd <- sqrt(var(tt) / n)

  result <- c(est, sd)
  names(result) <- c("estimate", "sd")
  return(result)
}
```

For $\alpha = 1.5$, $\Gamma(\alpha) = 0.8862269$. And for different simulation sizes, we have:

```
##       n  estimate          sd
## 1    10 0.8089427 0.133440121
## 2   100 0.8752116 0.046489115
## 3  1000 0.8913727 0.014615351
## 4 10000 0.8872262 0.004663275
```

d)

```
estimate_gamma_antithetic <- function(alpha, n = 1000) {
  u <- runif(n)
  u2 <- 1 - u
  u <- c(u, u2)
  ee <- -log(1-u)

  tt <- ee^(alpha - 1)
  est <- mean(tt)
  sd <- sqrt(var(tt) / (2*n))

  result <- c(est, sd)
  names(result) <- c("estimate", "sd")
  return(result)
}
```

We can see that the method of antithetic variates can reduce the standard error.

```
ttd.df
```

```
##    estimate          sd       type
## 1 0.8863899 0.004638395         MC
## 2 0.8856987 0.003260920 Antithetic
```

e)

```
tr_val <- 1 - exp(-1)
n <- 100000
```

i)

```r
u <- runif(n)
tt <- as.numeric(u < exp(-(1:n) / n))
c(mean(tt), sqrt(var(tt) / n))
```

```
## [1] 0.631020000 0.001525897
```

ii)

```r
u <- runif(n)
tt <- exp(-u)
c(mean(tt), sqrt(var(tt) / n))
```

```
## [1] 0.6319321185 0.0005718087
```

iii)

```r
u <- runif(n)
u2 <- 1 - u
u <- c(u, u2)
tt <- exp(-u)
c(mean(tt), sqrt(var(tt) / (2*n)))
```

```
## [1] 0.6322122636 0.0004054393
```

iv)

We performed a grid search on $(0,5) \times (0,5)$, and found that $\alpha = 0.8$ and $\beta = 1.2$ gives the smallest standard error.

```r
is_beta <- function(a=2, b=2, n=1000) {
  bb <- rbeta(n, a, b)
  tt <- exp(-bb) / dbeta(bb, a, b)
  est <- mean(tt)
  sd <- sqrt(var(tt) / n)
  result <- c(est, sd)
  names(result) <- c("estimate", "sd")
  return(result)
}

grid <- readRDS("grid.rds")
grid$sd %>% which.min() %>% {grid[.,]}
```

```
## # A tibble: 1 x 5
##       a     b estimate       sd     diff
##   <dbl> <dbl>    <dbl>    <dbl>    <dbl>
## 1   0.8   1.2    0.632 0.000255 0.000317
```

```r
# grid <- expand.grid(a=seq(0.2,5,by=.2), b=seq(0.2,5,by=.2))
#
# grid %<>% as_tibble %>% mutate(
#   is_beta_result = purrr::map2(a, b, function(aa, bb) {
#     rr <- is_beta(aa, bb, n = 100000) %>% t %>% as_tibble
#     colnames(rr) <- c("estimate", "sd")
#     rr
#   })
# ) %>% unnest(is_beta_result) %>%
#   mutate(
#     diff = abs(tr_val - estimate)
#   )
#
# grid$sd %>% which.min() %>% {grid[.,]}
# grid$diff %>% which.min() %>% {grid[.,]}
# grid %>% filter(a==1, b==1)
# is_beta(1, 1, n = 100000)
#
# saveRDS(grid, "homework1/grid.rds")
```

We can can see that from i to iv, the variability of methods decreases in terms of the standard errors. And the comparison of the time is shown below:

```
##                  test replications elapsed relative
## 1         hit-or-miss           10   0.459    1.000
## 2            basic MC           10   0.481    1.048
## 4 importance sampling           10   0.757    1.649
## 3          antithetic           10   0.956    2.083
```

Q2

a)

The maximum likelihood estimator of the shape parameter is:

```r
library(boot)
require(fitdistrplus)
```

```
## Loading required package: fitdistrplus
```

```
## Loading required package: MASS
```

```
##
## Attaching package: 'MASS'
```

```
## The following object is masked from 'package:dplyr':
##
##     select
```

```
## Loading required package: survival
```

```
##
## Attaching package: 'survival'

## The following object is masked from 'package:boot':
##
##     aml
```

```
data("aircondit")

tb.find <- function(alpha, data) {
  xbar <- mean(data)
  result <- log(alpha) - log(xbar) - digamma(alpha) + mean(log(data))
  return(result)
}

alpha.est.result <- uniroot(tb.find, interval = c(0.001, 4), data = aircondit$hours)
alpha.est <- alpha.est.result$root
alpha.est
```

```
## [1] 0.7064924
```

b)

The bias and variance of the shape parameter estimator obtained by the jackknife is:

```
jk.est <- sapply(seq_along(aircondit$hours), function(i) {
  data <- aircondit$hours
  data.tmp <- data[-i]

  est <- uniroot(tb.find, interval = c(0.001, 4), data = data.tmp)
  est$root
})
n <- length(aircondit$hours)
eiv <- (n-1) * (alpha.est - jk.est)

bias.alpha <- - mean(eiv)
var.alpha <- (sum(eiv^2) - n * bias.alpha^2) / (n * (n-1))

result <- c(bias.alpha, var.alpha)
names(result) <- c("bias", "variance")
result
```

```
##       bias   variance
## 0.15015204 0.06605922
```

Q3.

(d) infinitesimal jackknife, regular jackknife and non-parametric bootstrap

i.

```
corr = cor(cd4$baseline, cd4$oneyear)
infjack_var <- function(x, y) {
  mux = mean(x)
  muy = mean(y)
  sx = sd(x)
  sy = sd(y)
  zx = (x - mux)/sx
  zy = (y - muy)/sy

  ifunc = zx * zy - 1/2 * cor(x, y) * (zx^2 + zy^2)
  return(sum(ifunc^2) * 1/(length(ifunc)^2))
}
infjack_var(cd4$baseline, cd4$oneyear)
```

```
## [1] 0.005701791
```

ii.

```
jackknife <- function(x, y) {
  est = c()
  for (i in 1:length(x)) {
    est = c(est, cor(x[-i], y[-i]))
  }
  corr = cor(x, y)
  bias = (length(x)  - 1) * (mean(est) - corr)
  n = length(x)
  tmp = (n - 1) *(corr - est)
  variance = (sum(tmp^2) - n * bias^2)/((n  - 1) * n)
  return(c(bias, variance))
}
jackknife(cd4$baseline, cd4$oneyear)
```

```
## [1] -0.006784288  0.008187035
```

iii.

```
boot <- function(x, y, b = 50) {
  n = length(x)
  est = c()
  for(i in 1:b) {
    s = sample(c(1:n), n, replace = TRUE)
    est = c(est, cor(x[s], y[s]))
```

```
  }
  corr = cor(x, y)
  bias = mean(est) - corr
  variance = var(est - corr)
  return(c(bias, variance))
}
boot(cd4$baseline, cd4$oneyear)
```

```
## [1] -0.02366386  0.01191796
```

(e).

percentile interval

```
bootci <- function(x, y, b = 1000) {
  n = length(x)
  est = c()
  for(i in 1:b) {
    s = sample(c(1:n), n, replace = TRUE)
    est = c(est, cor(x[s], y[s]))
  }
  return(quantile(est, probs = c(0.025, 0.975)))
}
bootci(cd4$baseline, cd4$oneyear)
```

```
##      2.5%     97.5%
## 0.5001352 0.8644203
```

normal approximation

```
bootci_norm <- function(x, y) {
  n = length(x)
  corr = cor(x, y)
  l = (1 + corr -(1- corr) * exp(2*1.96/sqrt(n-3)))/(1 + corr +(1- corr)*exp(2*1.96/sqrt(n-3)))
  r = (1 + corr -(1- corr) * exp(-2*1.96/sqrt(n-3)))/(1 + corr +(1- corr)*exp(-2*1.96/sqrt(n-3)))
  return(c(l, r))
}

bootci_norm(cd4$baseline, cd4$oneyear)
```

```
## [1] 0.4127149 0.8830870
```

invariance to transformation

```r
fishertans <- function(x, y) {
  n = length(x)
  corr = cor(x, y)

  z = 0.5 * log((1 + corr)/(1-corr))
  sd = sqrt(1/(n-3))
  l = z + qnorm(0.975) * sd
  r = z - qnorm(0.975) * sd
  up = (exp(2 * l) - 1)/(exp(2*l) + 1)
  lo = (exp(2 * r) - 1)/(exp(2*r) + 1)
  return(c(lo, up))
}
fishertans(cd4$baseline, cd4$oneyear)
```

```
## [1] 0.4127222 0.8830851
```

Q4. (b).

CI Interval example:

bootstrap normal: 0.62 2.78

bootstrap basic: 0.29 1.82

bootstrap studentized: 1.37 3.41

Based on 1600 independent data sets:

Coverage:

bootstrap normal: 0.99875

bootstrap basic: 0.935625

bootstrap studentized: 0.94375

Mean and sd of length:

bootstrap normal: 2.3482 +/- 0.977

bootstrap basic: 2.51 +/- 0.71

bootstrap studentized: 2.61 +/- 1.3

The studentized bootstrap method was closest to 95% and bootstrap normal tends to give far large interval.

Below is the implementation

1. normal

```
normalci <- function(n = 15, theta = 2, r = 999, a = 0.025) {
  obs = rexp(n, theta)
  xbar = rep(0, r)
  for(i in 1:r) {
    x=sample(obs, n, replace=T)
    xbar[i] = 1/mean(x)
  }
  deltastar = xbar - theta
  d = quantile(deltastar, c(a,1-a))
  ci = mean(xbar) - c(d[2], d[1])
  return(ci)
}
normalci()
```

```
n = 15
theta = 2
r = 999
a = 0.025
sims <- 1600
results <- as.numeric(sims)
len = rep(0, sims)
for (l in 1:1600) {
  pop <- rexp(100, theta)
  mu <- 1/mean(pop)
  obs = sample(pop, n)
  xbar = rep(0, r)
  for(i in 1:r) {
    x=sample(obs, n, replace=T)
    xbar[i] = 1/mean(x)
  }
  deltastar = xbar - theta
  d = quantile(deltastar, c(a,1-a))
  ci = mean(xbar) - c(d[2], d[1])
  len[l] = ci[2] - ci[1]
  results[l] <- ci[1] < mu & ci[2] > mu
}
sum(results)/sims
```

2. basic

```
basicci <- function(n = 15, theta = 2, r = 999, a = 0.025) {
  obs = rexp(n, theta)
  est = 1/mean(obs)
  xbar = rep(0, r)
  for(i in 1:r) {
    x = rexp(n, est)
    xbar[i] = 1/mean(x)
  }
  xbar = sort(xbar)
  l = (r + 1) * (1 - a)
  u = (r + 1) * (a)
```

```
  return(c(2*est - xbar[l], 2*est - xbar[u]))
}
basicci()
```

```
results <- 0
lens = rep(0, sims)
for (j in 1:1600) {
  pop <- rexp(100, theta)
  mu <- 1/mean(pop)
  obs = sample(pop, n)
  est = 1/mean(obs)
  xbar = rep(0, r)
  for(i in 1:r) {
    x = rexp(n, est)
    xbar[i] = 1/mean(x)
  }
  xbar = sort(xbar)
  l = (r + 1) * (1 - a)
  u = (r + 1) * (a)
  ci = c(2*est - xbar[l], 2*est - xbar[u])
  lens[j] =  xbar[l] -  xbar[u]
  if(ci[1] < mu & ci[2] > mu)
    results = results + 1
}
results/sims
```

3. studentized (found 2 ways to implement this)

```
stuci <- function(n = 15, theta = 2, r = 999, a = 0.025, M = 50) {
  obs = rexp(n, theta)
  est = 1/mean(obs)
  zr = rep(0, r)
  xbar = rep(0, r)
  for(i in 1:r) {
    x=sample(obs, n, replace=T)
    xbar[i] = 1/mean(x)
    thetahat = rep(0, M)
    for (j in 1:M) {
      sx=sample(x, n, replace=T)
      thetahat[j] = 1/mean(sx)
    }
    vr = sd(thetahat)
    zr[i] = (xbar[i] - est)/vr
  }
  v = sd(xbar)
  zr = sort(zr)
  d = quantile(zr, c(a,1-a))
  ci = est - c(d[2], d[1])
  return(ci)
}
stuci()
```

```r
M = 10
results <- 0
for (l in 1:1600) {
  pop <- rexp(100, theta)
  mu <- 1/mean(pop)
  obs = sample(pop, n)
  est = 1/mean(obs)
  zr = rep(0, r)
  xbar = rep(0, r)
  for(i in 1:r) {
    x=sample(obs, n, replace=T)
    xbar[i] = 1/mean(x)
    thetahat = rep(0, M)
    for (j in 1:M) {
      sx=sample(x, n, replace=T)
      thetahat[j] = 1/mean(sx)
    }
    vr = sd(thetahat)
    zr[i] = (xbar[i] - est)/vr
  }
  v = sd(xbar)
  zr = sort(zr)
  d = quantile(zr, c(a,1-a))
  ci = est - c(d[2], d[1])
  if(ci[1] < mu & ci[2] > mu)
    results = results + 1
}
sum(results)/sims
```

```r
## or use (another implementation)
pop <- rexp(1000, rate = 1/theta)
x <- sample(pop, n)
mu <- mean(pop)
B <- 1000
samples <- matrix(sample(x, size = n * B, replace = T), nrow = n, ncol = B)
est <- (apply(samples, 2, mean) - mean(x))/(apply(samples, 2, sd)/sqrt(n))
mean(x) - quantile(est, probs = c(1-a, a)) * sd(x)/sqrt(n)
```

```r
ci <- matrix(FALSE, sims, 2)
results <- as.numeric(sims)
lens = rep(0, sims)
for (i in 1:sims) {
    pop <- rexp(1000, rate = 1/theta)
    x <- sample(pop, n)
    mu <- mean(pop)
    B <- 1000
    samples <- matrix(sample(x, size = n * B, replace = T), nrow = n, ncol = B)
    est <- (apply(samples, 2, mean) - mean(x))/(apply(samples, 2, sd)/sqrt(n))
    ci[i, ] <- mean(x) - quantile(est, probs = c(1-a, a)) * sd(x)/sqrt(n)
    lens[i] = ci[i, 2] - ci[i, 1]
    results[i] <- ci[i, 1] < mu & ci[i, 2] > mu
}
sum(results)/sims
```