STAT 428: Homework 4: Chapter 6 Monte Carlo Methods in Inference

Du, Yuting, yutingd3 Collaborated with: Sun, Yifan, yifans8

Table of Contents

Please refer to the [**detailed homework policy document**] on Course Page for information about homework formatting, submission, and grading.

Exercise 1

An Exploration of Standard Error in Monte Carlo Estimation

Consider the following integral:

$$\int_0^1 \frac{\ln(x+1)}{\pi\sqrt{x(1-x)}}\,dx.$$

a. Estimate the integral using naive Monte Carlo. What is the standard error of this estimate?

```
u = runif(1000)
I0 = mean(log(u+1)/(pi*sqrt(u*(1-u))))
I0

## [1] 0.3844
```

```r
x = numeric()
for(i in 1:1000){
  x[i] = mean(log(u[i]+1)/(pi*sqrt(u[i]*(1-u[i]))))
}
E0 = sd(x)/sqrt(1000)
E0

## [1] 0.02287
```

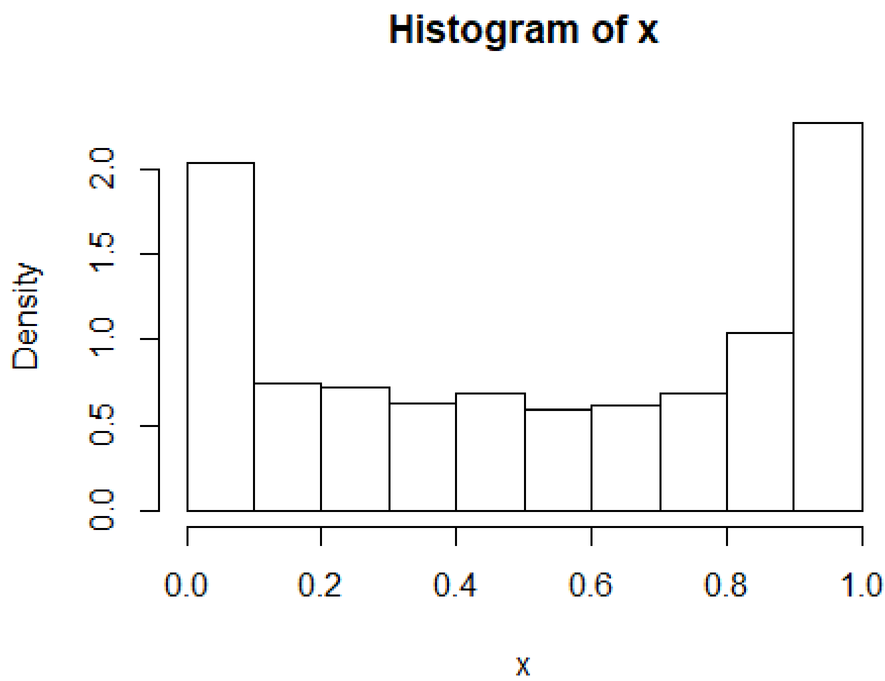
b. Let's see if we can improve the standard error. Implement Monte Carlo with antithetic sampling to estimate this integral. What is the standard error of this estimate?

```r
I1 = numeric()
I11 = numeric()
I12 = numeric()


for(i in 1:1000){
  k1 <- 500
  u1 <- runif(500)
  I11[i] <-  mean(log(u[i]+1)/(pi*sqrt(u[i]*(1-u[i]))))

  k2 <- 500
  u2 <- runif(500)
  I12[i] <-  mean(log(1-u[i]+1)/(pi*sqrt(u[i]*(1-u[i]))))


  I1[i] <- (I11[i] + I12[i])/2
}

E1 <- sd(I1)/sqrt(1000)
E1

## [1] 0.01227
```

c. Would stratified sampling seem to help here? Why or why not? (Whatever you decide, you do not need to implement it).

I believe that stratified sampling will also work in reducing the standard error. It uses simaple random sampling from uniform distributions but stratify these to ensure balance over a partition into k subintervals of (0, 1)

d. $f(x) = \frac{1}{\pi\sqrt{x(1-x)}}$ for $x \in (0,1)$ is the probability density function for the Arcsine distribution. Using inverse transformation method, sample 1000 random values from the Arcsine distribution.



e. Use importance sampling and the code you wrote in part d to estimate this integral. What is the standard error?
The importance function we choose is the pdf of Arcsine distribution.

```
m = 1000

gf = function(x){
  g = log(x+1)/(pi*sqrt(x*(1-x)))
  f = (x<1)*(x>0)
  g*f
}

##try our importance function
u = runif(m)
x = (sin(pi*u/2))^2
gfphi = gf(x)*(pi*sqrt(x*(1-x)))
E2 = sd(gfphi)/sqrt(m)
E2

## [1] 0.007742
```

f. Are all methods equally effective? Which method is the most efficient?

No. It seems like importance sampling is the most efficient way to reduce the error variance.

Exercise 2

Comparing MSE of estimators using MC.

Let $f(x|\theta) = \text{t}(\nu, \mu)$, the non-central t-distribution, where μ is a location parameter and ν is the degrees of freedom.

Estimate the MSE of the level k trimmed means for random samples of size 20 generated from a a non-central t-distribution with degrees of freedom 3 and mean 4 (with $\nu = 3$ and $\mu = 4$). Summarize the estimates of MSE in a table for k = 1, 2, … , 9.

```
n = 20
K = n/2 - 1
m = 1000
mse = matrix(0, n/2, 2)

trimmed.mse = function(n, m, k){
  tmean = numeric(m)
  for(i in 1:m){
    x = sort(rt(m, 3, 4))
    tmean[i] = mean(x[(k+1):(n-k)])
  }
  mse.est = mean(tmean^2)
  se.mse = sqrt(mean((tmean - mean(tmean))^2))/sqrt(m)
  return (c(mse.est, se.mse))
}

for(k in 1:K){
  mse[k, 1:2] = trimmed.mse(n,m,k)
}

round(mse, 3)
```

```
##        [,1]  [,2]
##  [1,] 1.885 0.003
##  [2,] 1.918 0.003
##  [3,] 1.931 0.003
##  [4,] 1.960 0.003
##  [5,] 1.983 0.003
##  [6,] 1.975 0.003
##  [7,] 1.995 0.003
##  [8,] 2.006 0.003
##  [9,] 2.016 0.003
## [10,] 0.000 0.000
```

Exercise 3

Bayesian Statistics Suppose $X_1, \dots, X_n$ are n independent and identical distributed random variables from $Exp(\theta)$, where θ is the unknown parameter. So,

$$f(x|\theta) = \theta\ e^{-\theta x}, \quad x \geq 0.$$

We assume the prior distribution on θ is the Gamma distribution ($Gamma(3,2)$).

$$g(\theta) = 4\theta^2 e^{-2\theta}, \quad x \geq 0.$$

1. Write down the posterior distribution of θ, $g(\theta|X)$.

$$g(\theta|x) = \frac{L(\theta)g(\theta)}{C}$$

where,

$$L(\theta) = \prod_{n=1}^{n} f\,(x_i|\theta)$$

and

$$g(\theta|x) = \int \prod_{n=1}^{n} f\,(x_i|\theta)g(\theta)dx$$

2. Suppose $n = 6$ and we observe that $x_1 = 0.4, x_2 = 1.1, x_3 = 0.2, x_4 = 1.6, x_5 = 1.4, x_6 = 0.9$. Estimate the posterior mean of θ based on 1000 simulated θ from its prior distribution.

```
observed_x = c(0.4, 1.1, 0.2, 1.6, 1.4, 0.9)

hx = numeric()
post.mean = numeric()
for(i in 1:length(observed_x)){
theta = rgamma(n = 1000, shape = 3, rate = 2)
hx[i] = theta*exp(-theta*observed_x[i])
c = mean(hx)
post.mean[i] = mean(theta*hx)/c
}
cbind(observed_x,post.mean)

##      observed_x post.mean
## [1,]        0.4     1.466
## [2,]        1.1     1.473
## [3,]        0.2     1.501
## [4,]        1.6     1.504
## [5,]        1.4     1.477
## [6,]        0.9     1.450
```

3. Suppose $n = 6$ and we observe that $x_1 = 0.4, x_2 = 1.1, x_3 = 0.2, x_4 = 1.6, x_5 = 1.4, x_6 = 0.9$.
    a. Design an acceptance-rejection sampling algorithm to generate 1000 (accepted) samples of θ from the posterior distribution of θ. Write down your algorithm with your instrumental distribution $g(\theta)$. (Hint: for the

acceptance-rejection sampling method, the normalizing constant in the posterior distribution can be ignored.)

Given that M = 0.5, for each observed x do folowing steps 1. generate 1 sample form gamma(3, 2) as the theta 2. generate 1 sample from uniform(1) 3. calculate f(x|theta) based on the observed x and theta from 1 4. calculate g(x) based on the observed x 5, Test uf the sample from 2 less than f(x|theta) / (M*g(x)) accept theta from 1 repeat until get 1000 accepted theta

b. Implement your acceptance-rejection sampling algorithm with R code. Plot the histogram of your generated sample and compare your sample mean with your estimated posterior mean obtained in Ex.3.2.

```
observed_x = c(0.4, 1.1, 0.2, 1.6, 1.4, 0.9)

fx = function(theta, x){
  theta*exp(-theta*x)*4*(theta^2)*exp(-2*theta)
}
AR = function(x, n){
  i = 0
  theta = numeric()
  M = 0.5
  while (i < n) {
    theta0 = rgamma(1,shape = 3, rate = 2)
    u = runif(1)
    if(M*u < fx(theta0, x)/dgamma(x, shape = 3, rate = 2)){
      i = i + 1
      theta[i] = theta0
    }
  }
  return(theta)
}
x = c(0.4,1.1,0.2,1.6,1.4,0.9)
AR(x = x, n = 6)

## [1] 0.7299 1.0390 0.4748 1.3879 1.3453 0.6148
```

Exercise 4

Do 6.1 in the book, except with $n = 25$ and $k = 1,2,\cdots,10$.

```
m <- 2000
K <- 10
n <- 25
tmean <- matrix(0,m,K)
mse_est <- numeric(K)
mse_se <- numeric(K)
for(k in 1:K){
  for (i in 1:m) {
    x <- sort(rcauchy(n))
    tmean[i,k] <- mean(x[(k+1):(n-k)])
```

```
    }
  mse_est[k] <- mean(tmean[,k]^2)
  mse_se[k] <- sqrt(sum((tmean[,k] - mean(tmean[,k]))^2)) / m
}
table <- cbind(seq(1:K),round(mse_est,5),round(mse_se,5))
colnames(table) <- c("k", "Estimated MSE of level k trimmed means",
"Standard Error")
knitr::kable(table, caption = 'Estimates of MSE')
```

Estimates of MSE

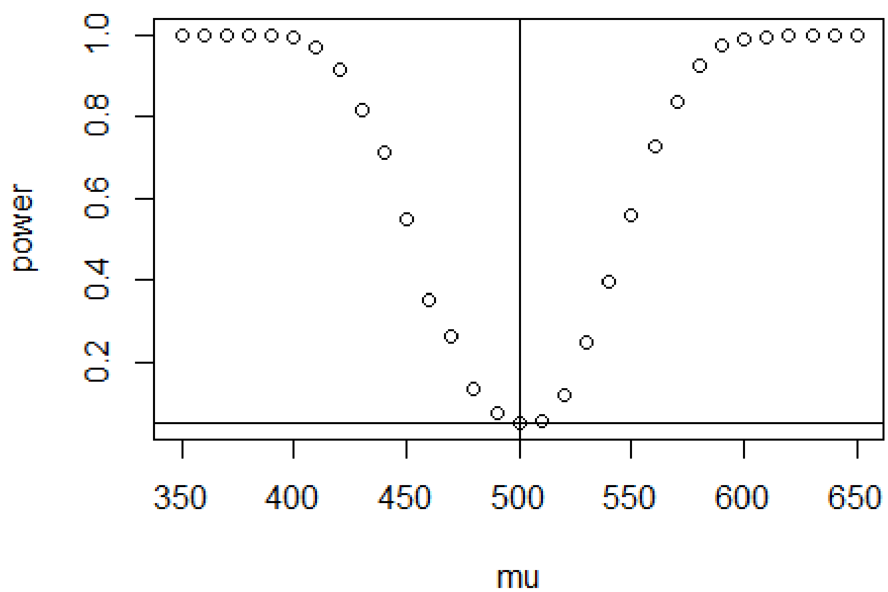
k	Estimated MSE of level k trimmed means	Standard Error
1	3.5117	0.0419
2	0.4430	0.0149
3	0.2132	0.0103
4	0.1696	0.0092
5	0.1418	0.0084
6	0.1244	0.0079
7	0.1159	0.0076
8	0.1092	0.0074
9	0.1026	0.0072
10	0.1066	0.0073

Exercise 5

Do exercise 6.2 from the book.

```
library(Hmisc) #for errbar

alpha = 0.05
mu <- c(seq(350, 650, 10)) #alternative H
n <- 20
sigma <- 100
m <- 1000
mu0 <- 500
M <- length(mu)
power <- numeric(M)
for (i in 1 : M) {
  pvalues <- replicate(m, expr = {x <- rnorm(n, mean = mu[i], sd =
sigma)
      ttest <- t.test(x, alternative = "two.sided", mu = mu0)
      ttest$p.value})
  power[i] <- mean(pvalues <= alpha)
}

plot(mu, power)
```

```
abline(v = mu0, lty = 1)
abline(h = alpha, lty = 1)
```

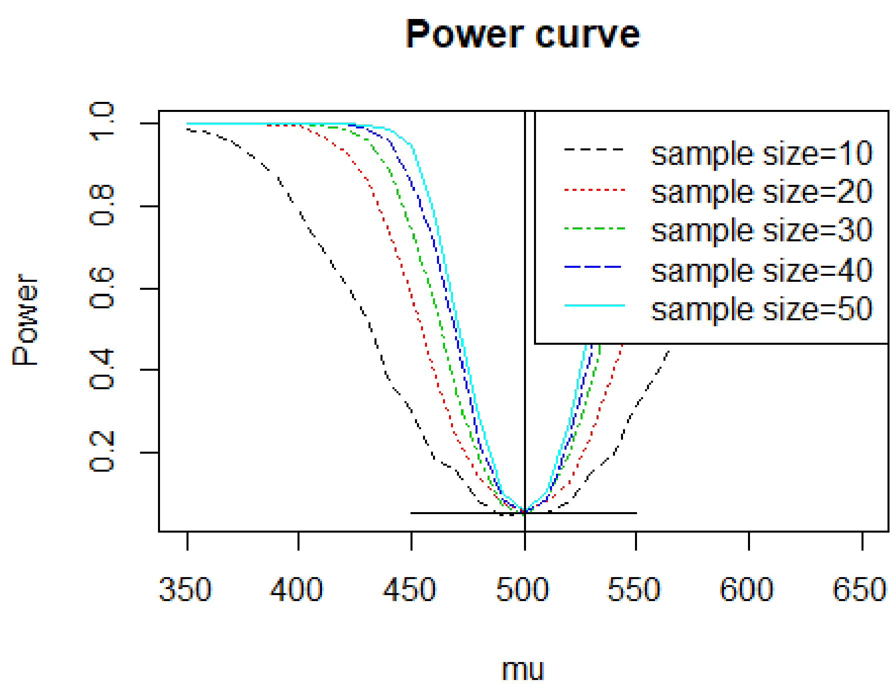


Exercise 6

Do exercise 6.3 from the book.

```
n <- seq(10,50,10) #sample size
mu <- c(seq(350, 650, 10))
m <- 1000
M <- length(mu)
N <- length(n)
power <- matrix(0,M,N)
for(j in 1:N){
  for (i in 1:M){
    mu1 <- mu[i]
    pvalues <- replicate(m, expr = {
      #simulate under alternative mu1
      x <- rnorm(n[j], mean = mu1, sd = 100)
      ttest <- t.test(x,
      alternative = "two.sided", mu = 500)
      ttest$p.value })
    power[i, j] <- mean(pvalues <= .05)
  }
}
plot(mu, power[,1], type="l", lty = 2, col = 1, main="Power curve",
xlab="mu", ylab="Power")
```

```
lines(mu, power[,2], lty = 3, col = 2)
lines(mu, power[,3], lty = 4, col = 3)
lines(mu, power[,4], lty = 5, col = 4)
lines(mu, power[,5], lty = 1, col = 5)
legend("topright", c("sample size=10","sample size=20","sample
size=30","sample size=40","sample size=50"), lty =
c(2,3,4,5,1),col=c(1,2,3,4,5))
abline(v = 500, lty = 1)
lines(c(450,550),c(0.05,0.05))
```



Exercise 7

Do exercise 6.5 from the book.

```
alpha <- 0.05
m <- 1000
n <- 20
qt <- qt(1-alpha/2, df = n-1)
LCL <- replicate(m, expr = {
  x <- rchisq(n, 2)
  return(mean(x) - qt * sd(x) / sqrt(n))
})
UCL <- replicate(m, expr = {
  x <- rchisq(n, 2)
  return(mean(x) + qt * sd(x) / sqrt(n))
})
mean((LCL < 2) * (UCL > 2))
```

```
## [1] 0.91
```

Exercise 8

Do exercise 6.8 from the book. Use 15 as small sample size, 50 as medium sample size, and 250 as large sample size.

```
alpha = 0.055
mu1 <- mu2 <- 0
sigma1 <- 1
sigma2 <- 1.5
sample_num <- c(15, 50, 250)

m <- 1000
tests_F <- numeric(3)
tests_CF <- numeric(3)

testF <- function(x, y) {
  f_test <- var.test(x, y, alternative = "two.sided", conf.level = 1-
alpha)
  return(f_test$p.value < alpha)
}
tests5 = function(X, Y) {
  outx <- sum(X > max(Y)) + sum(X < min(Y))
  outy <- sum(Y > max(X)) + sum(Y < min(X))
  return(as.integer(max(c(outx, outy)) > 5))
}
for (i in 1 : 3) {
  n <- sample_num[i]
  power_F <- mean(replicate(m, expr = {
  x <- rnorm(n, mu1, sigma1)
  y <- rnorm(n, mu2, sigma2)
  testF(x, y)
}))
  power_CF <- mean(replicate(m, expr = {
  x <- rnorm(n, mu1, sigma1)
  y <- rnorm(n, mu2, sigma2)
  tests5(x, y)
}))

  tests_F[i] <- power_F
  tests_CF[i] <- power_CF
}

(table = rbind(tests_F, tests_CF))

##            [,1]  [,2]  [,3]
## tests_F  0.321 0.800 1.000
## tests_CF 0.290 0.655 0.973
```