PSTAT 194CS HW 4
Monte Carlo Methods for Inference

Ostapenko, Vasiliy (vostapenko, 774 970 8)
Collaborated with: N/A

Contents

Exercise 1 1
Exercise 2 2
Exercise 3 3
Exercise 4 (6.1) 4
Exercise 5 (6.2) 5
Exercise 6 (6.3) 6
Exercise 7 (6.5) 7
Exercise 8 (6.8) 7

Exercise 1

```r
mc_sim_1 = function(n=1000, k=1000, a=0, b=1, anti=FALSE, g=identity) {
  x = vector(mode="numeric", length=n)
  for(i in 1:n) {
    if(anti == TRUE) {
      u_vec = 1 - runif(n=k, min=a, max=b)
    }
    else {
      u_vec = runif(n=k, min=a, max=b)
    }
    u_vec = g(u_vec)
    h_vec = (log(u_vec+1))/(pi*sqrt(u_vec*(1-u_vec)))
    x[i] = (b-a) * (1/k) * sum(h_vec)
  }
  return(x)
}
```

1a.

```r
est = mc_sim_1(n=1, k=10000, a=0, b=1, anti=FALSE, g=identity)
se = mc_sim_1(n=1000, k=10000, a=0, b=1, anti=FALSE, g=identity) %>% sd()
print(paste0("Estimate: ", round(est, 3), "; Std. Error: ", round(se, 3)))
```

```
## [1] "Estimate: 0.374; Std. Error: 0.008"
```

1b.

```r
est = mc_sim_1(n=1, k=10000, a=0, b=1, anti=TRUE, g=identity)
se = mc_sim_1(n=1000, k=10000, a=0, b=1, anti=TRUE, g=identity) %>% sd()
print(paste0("Estimate: ", round(est, 3), "; Std. Error: ", round(se, 3)))
```

```
## [1] "Estimate: 0.378; Std. Error: 0.009"
```

1c. Stratified sampling might help our MC estimate in this case. Across the interval (0, 1), the function is not constant and increases its rate of change as it gets closer to 1. Thus, that region might be split from the rest and sampled from separately.

1d.

```
cdfInv = function(x) {
  return(1-(sin(pi*x/2))^2)
}
```

```
u = runif(n=1000, min=0, max=1)
sample = cdfInv(u)
print(sample[1:10])
```

```
##  [1] 0.70330 0.93333 0.24190 0.06413 0.95830 0.18306 0.17423 0.07854 0.20937
## [10] 0.93953
```

1e.

```
est = mc_sim_1(n=1, k=10000, a=0, b=1, anti=FALSE, g=cdfInv)
se = mc_sim_1(n=1000, k=10000, a=0, b=1, anti=FALSE, g=cdfInv) %>% sd()
print(paste0("Estimate: ", round(est, 3), "; Std. Error: ", round(se, 3)))
```

```
## [1] "Estimate: 1.764; Std. Error: 10.393"
```

1f. The methods are not equally effective. The method which was most efficient was the naive sampling MC method. Using importance sampling with the arcsine density, our MC estimate of the integral was off by about 400%, with a standard error which was higher by a factor of one thousand.

Exercise 2

```
trimmed_sim_1 = function(k=9, m=1000, n=20) {
  x = list()
  for(i in 1:k) {
    tmean = vector(mode="numeric", length=m)
    for(j in 1:m) {
      s = rt(n=n, df=3, ncp=4)
      s = sort(s)
      tmean[j] = mean( s[(i+1) : (n-i)] )
    }
    mse.est = mean(tmean^2)
    se.mse = sqrt(mean((tmean - mean(tmean))^2))/sqrt(m)

    x[[i]] = list(k=i, `t-mean`=mse.est, se=se.mse)
  }
  x = rbindlist(x)
  return(x)
}

x = trimmed_sim_1(k=9, m=1000, n=20)
print(x)
```

```
##    k t-mean      se
## 1: 1  26.65 0.02286
## 2: 2  24.56 0.02082
## 3: 3  23.63 0.02072
## 4: 4  22.44 0.01900
## 5: 5  22.19 0.01959
## 6: 6  21.59 0.01956
## 7: 7  20.96 0.01960
## 8: 8  20.92 0.01905
## 9: 9  20.74 0.01973
```

Exercise 3

1. The posterior distribution of theta, g(theta | X), is as follows:

$$
\begin{aligned}
g(\theta|\mathbf{x}) &\propto f(\mathbf{x}|\theta)g(\theta) && (1)\\
&= \theta^n e^{-\theta\sum_{i=1}^n x_i} 4\theta^2 e^{-2\theta} && (2)\\
&\propto \theta^{n+2} e^{-\theta(2+\sum_{i=1}^n x_i)} && (3)
\end{aligned}
$$

This is proportional to a Gamma distribution, thus $\theta|\mathbf{x} \sim Gamma(n+3, 2+\sum_{i=1}^n x_i)$, and

$$g(\theta|\mathbf{x}) = \frac{(2+\sum_{i=1}^n x_i)^{n+3}}{\Gamma(n+3)}\theta^{n+2}e^{-\theta(2+\sum_{i=1}^n x_i)}$$

2.

```r
x = c(0.4, 1.1, 0.2, 1.6, 1.4, 0.9)
theta = rgamma(n=10000, shape=3, rate=2)
l = (theta^length(x))*exp(-theta*sum(x))
mean(theta*l)/mean(l)
```

```
## [1] 1.181
```

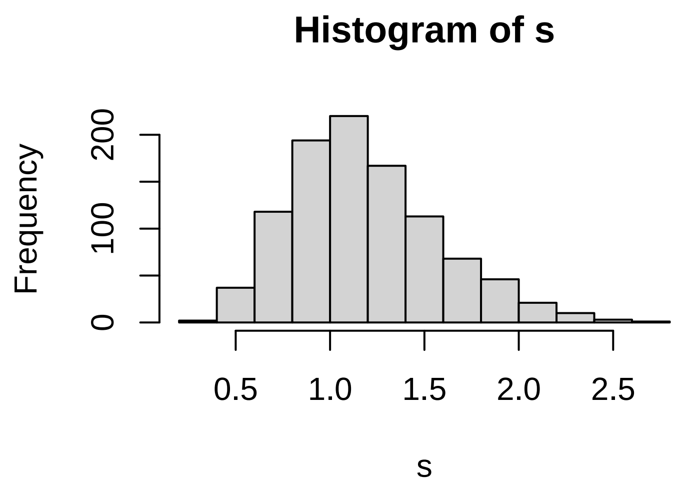
3a. Algorithm for AR method: - 1. Find M: since $g(\theta|\mathbf{x}) \propto f(\mathbf{x}|\theta)g(\theta)$, then $M = sup_\theta \frac{g(\theta|\mathbf{x})}{g(\theta)} = sup_\theta f(\mathbf{x}|\theta)$ - 2. Draw a sample point θ from prior distribution $g(\theta)$ - 3. Generate a random number $u \in [0,1]$ - 4. Decide whether accept θ from step 1 as a sample point from $g(\theta|\mathbf{x})$ by checking condition $u < \frac{f(\mathbf{x}|\theta)}{M}$; Yes—accept, No—reject - 5. Repeat steps 2-3 until 1000 values are generated

3b.

```r
likelihood = function(theta, x) {
  return(theta^length(x) * exp(-theta*sum(x)))
}

i = 1
s = vector(mode="numeric", length=1000)
M = optimize(likelihood, c(0,5), maximum=TRUE, x=x)$objective

while(i <= 1000) {
  theta = rgamma(1, 3, 2)
  u = runif(1, 0, 1)
  if(u < likelihood(theta, x)/M) {
    s[i] = theta
    i = i+1
  }
}

mean(s)
```

```
## [1] 1.187
```

```
hist(s)
```



Exercise 4 (6.1)

Use n=25; k=1:10

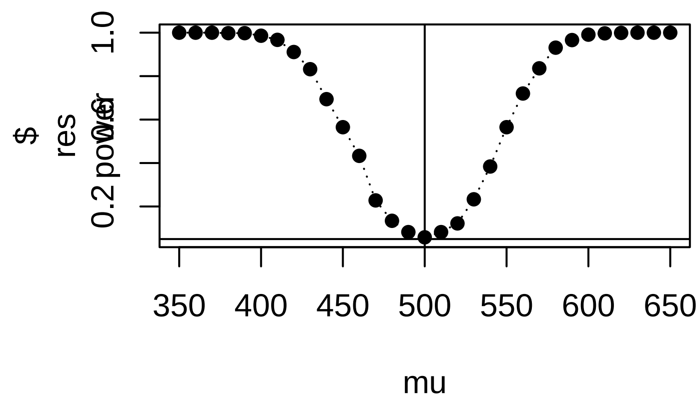
```
trimmed_sim_2 = function(k=9, m=1000, n=20) {
  x = list()
  for(i in 1:k) {
    tmean = vector(mode="numeric", length=m)
    for(j in 1:m) {
      s = rcauchy(n=n)
      s = sort(s)
      tmean[j] = mean( s[(i+1) : (n-i)] )
    }
    mse.est = mean(tmean^2)
    se.mse = sqrt(mean((tmean - mean(tmean))^2))/sqrt(m)

    x[[i]] = list(k=i, `t-mean`=mse.est, se=se.mse)
  }
  x = rbindlist(x)
  return(x)
}

x = trimmed_sim_2(k=10, m=1000, n=25)
print(x)
```

```
##      k  t-mean       se
##  1:  1 1.32500 0.036346
##  2:  2 0.60076 0.024509
##  3:  3 0.22253 0.014916
##  4:  4 0.17016 0.013027
##  5:  5 0.14825 0.012175
##  6:  6 0.13552 0.011640
##  7:  7 0.10014 0.010007
##  8:  8 0.10332 0.010151
##  9:  9 0.09762 0.009874
## 10: 10 0.10359 0.010177
```

Exercise 5 (6.2)

Plot empirical power curve for t-test with H0: mu = 500; H1: mu != 500; alpha = 0.05.

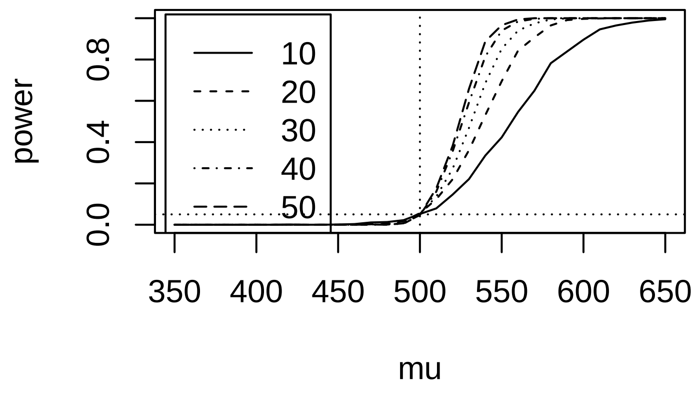
```r
power_sim_1 = function(mu_low=350, mu_high=650, m=1000, n=20, sigma=100,
                       twosided=TRUE) {
  if(twosided) {
    alt = "two.sided"
  } else {
    alt = "greater"
  }
  mu = seq(mu_low, mu_high, 10)
  power = vector(mode="numeric", length=length(mu))
  for(i in 1:length(mu)) {
    mu_temp = mu[i]
    pvals = vector(mode="numeric", length=m)
    for(j in 1:m) {
      x = rnorm(n=n, mean=mu_temp, sd=sigma)
      ttest = t.test(x, alternative=alt, mu=500)
      pvals[j] = ttest$p.value
    }
    power[i] = mean(pvals <= 0.05)
  }
  return(list(mu=mu, power=power))
}

res = power_sim_1(350, 650, 1000, 20, 100, TRUE)

{
  se = sqrt(res$power*(1-res$power)/1000)
  errbar(res$mu, res$power, yplus=res$power+se,
         yminus=res$power-se, xlab="mu")
  abline(v=500, lty=1)
  abline(h=0.05, lty=1)
  lines(res$mu, res$power, lty=3)
}
```



Exercise 6 (6.3)

```r
curves = list()
sizes = seq(10, 50, 10)
for(i in 1:length(sizes)) {
  temp_res = power_sim_1(350, 650, 1000, sizes[i], 100, FALSE)
  temp_pow = temp_res$power
  curves[[i]] = list(`size`=sizes[i], `pow`=temp_pow)
}
```

```r
x = seq(350, 650, 10)
{
  plot(x, curves[[1]]$pow, type="l", ylim=c(0, 1), xlab="mu", ylab="power")
  abline(v=500, lty=3)
  abline(h=0.05, lty=3)
  for(i in 2:length(curves)) {
    lines(x, curves[[i]]$pow, lty=i)
  }
  legend("topleft", inset=0.02, legend=sizes, lty=1:5)
}
```



Exercise 7 (6.5)

```r
n = 20
t0 = qt(c(0.025, 0.975), df=n-1)
CI = replicate(10000, expr={
  x = rchisq(n, df=2)
  ci = mean(x)+t0*sd(x)/sqrt(n)
  })

low = CI[1, ]
high = CI[2, ]
print(sum(low < 2 & high > 2))
```

```
## [1] 9195
```

```r
print(mean(low < 2 & high > 2))
```

```
## [1] 0.9195
```

Exercise 8 (6.8)

Use samples of size 15, 50, and 250

```r
count5test = function(x, y) {
  X = x - mean(x)
  Y = y - mean(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))
}

power_sim_2 = function(samples=c(15, 50, 250), m=10000) {
  x = list()
  i = 1
  for (n in samples) {
    tests = replicate(m, expr={
      x = rnorm(n, 0, 1)
      y = rnorm(n, 0, 1.5)
      C5 = count5test(x, y)
      Fp = var.test(x, y)$p.value
      Ftest = as.integer(Fp <= 0.055)
      c(C5, Ftest)
    })
    rm = rowMeans(tests)
    x[[i]] = list(`n`=n, `low`=rm[[1]], `high`=rm[[2]])
    i = i+1
  }
  x = rbindlist(x)
  return(x)
}

res = power_sim_2()
print(res)
```

```
##      n    low   high
## 1:  15 0.2115 0.3122
## 2:  50 0.6637 0.8131
## 3: 250 0.9633 1.0000
```