HW2

Yan Liu

9/15/2019

1.

a)

Let X be an exponential r.v. with rate 1. Using cdf inversion, write a function that generates n independent samples of X (we discussed this example in class, but you should do the cdf inversion yourself, not just quote our result). Compare the speed of your sampler for $n = 10^6$ with that of your language's exponential sampler (in R **rexp**).

Solution:

Since $X \sim Exp(1)$, we have PDF $f(x) = e^{-x}$, $x \in [0, \infty)$, and CDF $F(x) = P(x \le c) = \int_0^c e^{-x} dx = 1 - e^{-c}$. Thus, $x = F(F^{-1}(x)) = 1 - e^{-F^{-1}(x)}$. Solve for $F^{-1}(x)$, we have $F^{-1}(x) = -ln(1 - x)$

```
time.start <- proc.time()
r.exp <- rexp(n=10^6)
proc.time() - time.start
```

```
##    user  system elapsed
##   0.043   0.002   0.045
```

```
myExpFunc <- function(N){
  return(-log(1-runif(N)))
}
time.start <- proc.time()
my.exp <- myExpFunc(N=10^6)
proc.time() - time.start
```

```
##    user  system elapsed
##   0.038   0.003   0.041
```

b)

Let X be a r.v. with the following pdf $f(x) = \frac{1}{2}\exp[-|x|]$, where $\exp[-|x|] = e^{-|x|}$. Write a function that samples from X using cdf inversion.Either theoretically or through numerical examples, show that your sampler correctly samples X.

Solution

$$f(x) = \frac{1}{2}\exp(-|x|) = \begin{cases} \frac{1}{2}e^{-x}, & \text{if } x \ge 0 \\ \frac{1}{2}e^{x}, & \text{otherwise} \end{cases}$$

Thus, when $c < 0$,

$$F(x) = \Pr(x \le c) = \int_{-\infty}^{c} \frac{1}{2}e^{x} dx = \frac{1}{2}e^{c}$$

When $c \geq 0$,

$$\begin{aligned} F(x) &= \Pr(x \leq c) \\ &= P(x < 0) + P(0 \leq x \leq c) \\ &= \int_{-\infty}^{0} \frac{1}{2} e^x dx \int_{0}^{c} \frac{1}{2} e^{-x} dx \\ &= \frac{1}{2} e^0 + (-\frac{1}{2} e^{-x}|_0^c) \\ &= 1 - \frac{1}{2} e^{-c} \end{aligned}$$

Thus,

$$F(x) = \begin{cases} \frac{1}{2}e^x, & \text{if } x < 0 \\ 1 - \frac{1}{2}e^{-x}, & \text{otherwise} \end{cases}$$

Let $y_1 = F(x)$ when $x < 0$, $y_2 = F(x)$ when $x \geq 0$.

$$\begin{aligned} y_1 &= \frac{1}{2} e^x \implies x = log(2y_1),\ where\ y_1 < \frac{1}{2} \\ y_2 &= 1 - \frac{1}{2} e^{-x} \implies x = -log(2 - 2y_2),\ where\ y_2 \geq \frac{1}{2} \end{aligned}$$

Thus,

$$F^{-1}(x) = \begin{cases} log(2x), & \text{if } x < \frac{1}{2} \\ -log(2-2x), & \text{otherwise} \end{cases}$$

Prove the two CDF are equivalent. Let C_x be any constant less than 0, y follows the distribution determined by F^{-1} above, $C_y = \frac{1}{2}e^{C_x} < \frac{1}{2}$, then,

$$\begin{aligned} F_y(C_y) &= \Pr(y \leq C_y) \\ &= \Pr(F_y^{-1}(y) \leq F_y^{-1}(C_y)) \\ &= \Pr(x \leq log(2 \times \frac{1}{2} e^{C_x}) \\ &= \Pr(x \leq C_x) \\ &= F_x(C_x) \end{aligned}$$

Let C_x be any constant equal to or greater than 0, $C_y = 1 - \frac{1}{2}e^{-C_x} \geq \frac{1}{2}$, then,

$$\begin{aligned} F_y(C_y) &= \Pr(y \leq C_y) \\ &= \Pr(F_y^{-1}(y) \leq F_y^{-1}(C_y)) \\ &= \Pr(x \leq -log(2 - 2 \times (1 - \frac{1}{2} e^{-C_x}))) \\ &= \Pr(x \leq -log(e^{-C_x})) \\ &= \Pr(x \leq C_x) \\ &= F_x(C_x) \end{aligned}$$

2.

Write a function **MarkovChain(P, s_0, s)** that simulates a Markov chain $X(t)$ until the first time the chain is in state s, assuming $X(0) = s_0$. The function should return the path of the chain from $t = 0$ to when it "hits" state s. You may use your language's discrete sampler (in R **sample**) or write your own.

```r
#' A helper function to determine the next state using runif()
#' @param row.cumsum cumsum probabilities of current row
#' @param states list of states
#' @returns next state
roll.dice <- function(row.cumsum, states){
  # roll a dice
  dice <- runif(1)
  n <- length(states)
  # could be the initial state
  if (dice >= 0 & dice < row.cumsum[1])
      return(states[1])
  else{
       for(i in seq(1,n)){
         # could be one of the middle states
         if (i <= n-1){
                  if (dice>= row.cumsum[i] & dice < row.cumsum[i+1]){
           return(states[i+1])
                  }
         }else{
           # otherwise the last state
             return(states[n])
           }
       }
  }
}
```

```r
#' A function MarkovChain(P, s0, s) that simulates a Markov chain X(t)
#' until the first time the chain is in state s, assuming X(0) = s0.
#' @param P a transition probability matrix
#' @param s0 initial state
#' @param s target state or end state
#' @details P must satisify: a square matrix; row sum must be 1;
#' each entry is less than or equal to 1;
#' the row names and the column names of P are both states.
#' The maximum steps to take is limited to 10000
#' @returns a list of states, i.e.,
#' the path of the chain from t = 0 to when it "hits" state s.
MarkovChain <- function(P,s0,s)
{
  if (is.null(s0))
    stop("Initial state must be non-null")

  if (is.null(s))
    stop("End state must be non-null")

  if (is.null(P)) {
    stop("Transition matrix must be non-null")
  }else{
    if (nrow(P) == ncol(P) & all(P <= 1) & all(P >= 0) & rowSums(P) == 1){
      path <- list()
      states <- row.names(P)
      s.current <- which(rownames(P)==s0)
      path[1] <- s0
      row.cumsum <- cumsum(P[s.current,])
```

```r
      s.next <- roll.dice(row.cumsum, states)
      numSteps <- 1
      maxSteps <- 10000
      while (s.next != s & numSteps <= maxSteps ){
        row.num.current <- which(rownames(P)==s.next)
        row.cumsum <- cumsum(P[row.num.current,])
        path[numSteps+1] <- s.next
        s.next <- roll.dice(row.cumsum, states)
        numSteps <- numSteps + 1
      }
      path[numSteps+1] <- s.next
    }
    else{
      stop("Transition matrix must be correctly specified")
    }
  }
  return (path)
}
```

```r
states <- c("worse","bad", "okay", "good","better")
P <- matrix(c(0,.2,.3,.4,.1,
              .5,.5,.0,0,0,
              .05,.05,.05,.05,.8,
              .5,.5,.0,0,0,
              0,.2,.3,.4,.1), nrow = 5, ncol = 5, byrow = TRUE,
              dimnames = list(states,states))
set.seed(10)
MarkovChain(P,"worse","better")
```

```
## [[1]]
## [1] "worse"
##
## [[2]]
## [1] "good"
##
## [[3]]
## [1] "worse"
##
## [[4]]
## [1] "okay"
##
## [[5]]
## [1] "better"
```

```r
set.seed(6)
MarkovChain(P,"good","better")
```

```
## [[1]]
## [1] "good"
##
## [[2]]
## [1] "bad"
##
## [[3]]
## [1] "bad"
```

```
##
## [[4]]
## [1] "worse"
##
## [[5]]
## [1] "okay"
##
## [[6]]
## [1] "better"
```

3.

Chutes and Ladders is a popular children's game. Use your function from problem 2 and a Monte Carlo approach to compute $E[L]$ where L is the average length, in terms of the number of die rolls, of a Chutes and Ladders game. Use a CLT argument to determine roughly how many games you have to simulate to estimate $E[L]$ to an accuracy of ± 5.

```
chute.ladder <- read.csv("chutes_and_ladder_locations.csv")
game.mx <- matrix(data = rep(0,100*100),
                  nrow = 100,
                  ncol = 100,
                  dimnames = list(seq(1,100),seq(1,100)))
for (i in 1:94){
  for (j in 1:6){
    game.mx[i, i+j] <- 1/6
  }
}

for (i in 95:99){
  j = 1
  while (j <= 99-i){
    game.mx[i,i+j] <- 1/6
    j <- j+1
  }
  game.mx[i,100] <- 1-sum(game.mx[i,1:99])
}
```

Take a glance

First few entries

```
game.mx[1:5,1:5]
```

```
##   1         2         3         4         5
## 1 0 0.1666667 0.1666667 0.1666667 0.1666667
## 2 0 0.0000000 0.1666667 0.1666667 0.1666667
## 3 0 0.0000000 0.0000000 0.1666667 0.1666667
## 4 0 0.0000000 0.0000000 0.0000000 0.1666667
## 5 0 0.0000000 0.0000000 0.0000000 0.0000000
```

Last few entries

```
game.mx[95:100,95:100]
```

```
##     95         96        97        98        99       100
## 95   0 0.1666667 0.1666667 0.1666667 0.1666667 0.3333333
## 96   0 0.0000000 0.1666667 0.1666667 0.1666667 0.5000000
## 97   0 0.0000000 0.0000000 0.1666667 0.1666667 0.6666667
## 98   0 0.0000000 0.0000000 0.0000000 0.1666667 0.8333333
## 99   0 0.0000000 0.0000000 0.0000000 0.0000000 1.0000000
## 100  0 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
```

Update based on chute and ladder locations

```
entrances <- chute.ladder$start
exits <- chute.ladder$end
update <- function(game.mx, entrances, exits){
    len <-  length(entrances)
    updated.mx <- game.mx
    for (i in seq(1:len)){
      updated.mx[entrances[i],] <- updated.mx[exits[i],]
    }
  return(updated.mx)
}
update.mx <- update(game.mx, entrances,exits)
```

Take a glance

First ladder

```
update.mx[c(1,38),35:40]
```

```
##    35 36 37 38        39        40
## 1   0  0  0  0 0.1666667 0.1666667
## 38  0  0  0  0 0.1666667 0.1666667
```

First chute

```
update.mx[c(98,78),80:85]
```

```
##           80        81        82        83        84 85
## 98 0.1666667 0.1666667 0.1666667 0.1666667 0.1666667  0
## 78 0.1666667 0.1666667 0.1666667 0.1666667 0.1666667  0
```

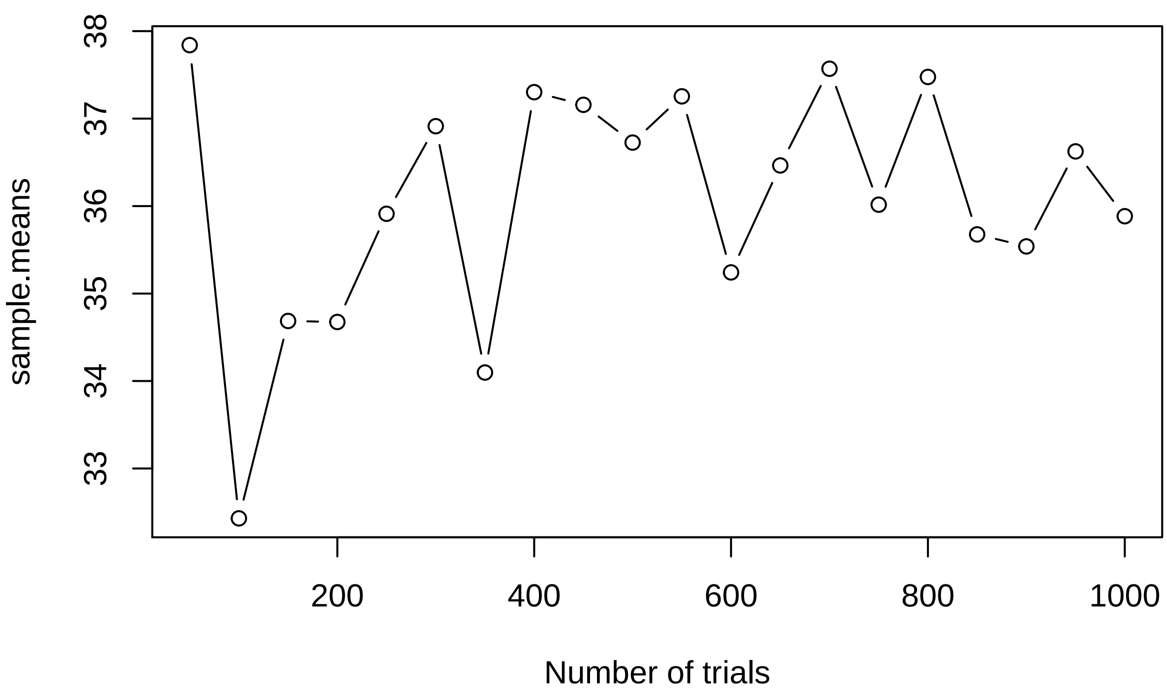
```
playGame <-  function(N){
  steps = numeric()
  for (i in 1:N){
    dice.0 <- runif(1, min = 0, max = 6)
    s0 <- case_when(
                    dice.0 <= 1~38,
                    dice.0 > 1 & dice.0 <= 2~2,
```

```r
                    dice.0 > 2 & dice.0 <= 3~3,
                    dice.0 > 3 & dice.0 <= 4~14,
                    dice.0 > 4 & dice.0 <= 5~5,
                    dice.0 > 5 & dice.0 <= 6~6
                    )

    path <- MarkovChain(update.mx, s0,100)
    steps[i] <- length(path)
  }
  return(steps)
}
set.seed(1024)
# number of times
Ns <- seq(50, 1000, 50)
sample.means <-  numeric()
sample.sigmas <- numeric()
for (i in 1:length(Ns)){
  steps <- playGame(Ns[i])
  sample.means[i] <- mean(steps)
  sample.sigmas[i] <- sqrt(var(steps))
}
```

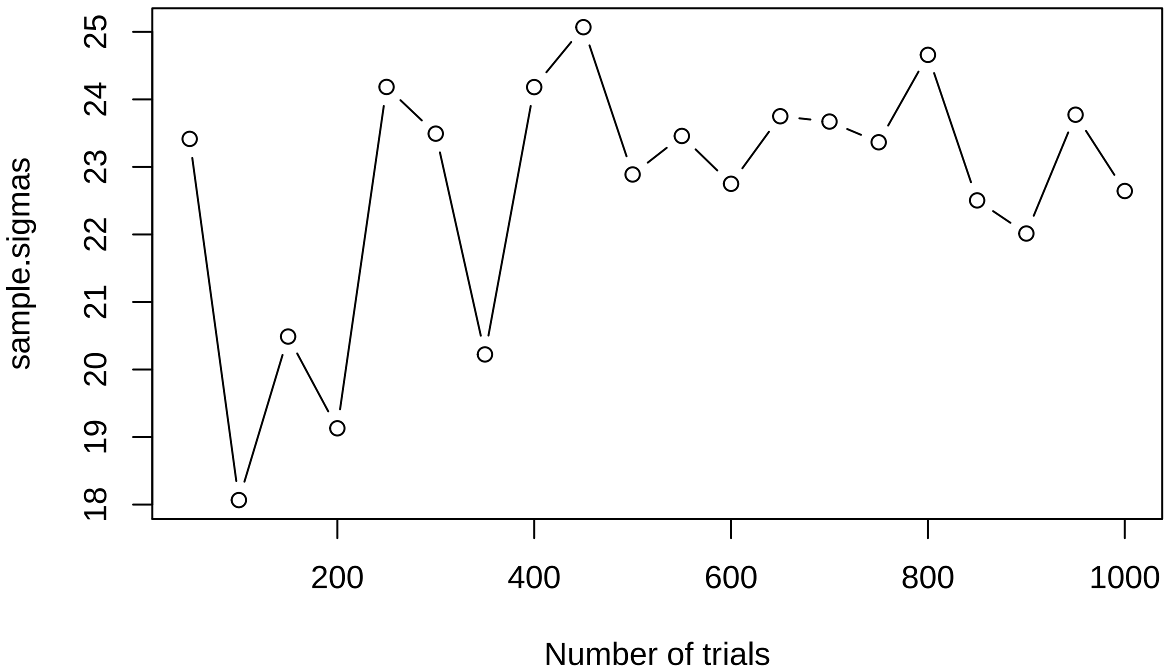
Check convergence of sample mean

```r
plot(Ns, sample.means, type = "b", xlab = "Number of trials")
```



Check convergence of sample standard deviation

```r
plot(Ns,sample.sigmas, type = "b", xlab = "Number of trials")
```



Next, determine the number of trials that would enable the estimated $\hat{E}[L]$ to have an accuracy of ± 5 compared with the true $E[L]$.Based on the central limit theorem,

$$\lim_{n\to\infty} \sqrt{n}(\frac{1}{n}\sum_{i=1}^{n}\hat{L}_i - E(L) \sim N(0,\sigma^2)$$

Let

$$\hat{E}[L] = \frac{1}{n}\sum_{i=1}^{n}\hat{L}_i$$

In other words,

$$\Pr(-5 \leq \hat{E}[L] - E[L] \leq 5) = \Pr(\hat{E}[L] - 5 \leq E[L] \leq \hat{E}[L] + 5)$$

After some transformations, we will have:

$$Pr(\frac{-5\sqrt{n}}{\sigma} \leq N(0,1) \leq \frac{5\sqrt{n}}{\sigma})$$

Assuming the confidence level is 99%, which corresponds to a Z-score of 2.58. Let $\frac{5\sqrt{n}}{\sigma} = 2.58$, where σ is estimated by $\hat{\sigma}$. According to the second plot above, $\hat{\sigma} \approx 24$. Solve the equation for n, we have $n \approx 153$.