HW1

Yigao Li

September 14, 2017

Problem 1

```
a <- floor(sin(1.23) * 1e10) - floor(sin(1.23) * 1e9) * 10
b <- sqrt(a^2 + a^3)
c <- nchar(floor(exp((log(b))^3)))
d <- 0
for (j in 1:c){
  d = d + j^3
}
e <- d %% 77
cat('a =', a, '\nb =', b, '\nc =', c, '\nd =', d, '\ne =', e)
```

```
## a = 9
## b = 28.4605
## c = 17
## d = 23409
## e = 1
```

Problem 2

```
library(tictoc)

mytoss = function(p){
  u <- runif(1)
  x <- as.numeric(u < p)
  return(x)
}

myattempts = function(p){
  counter <- 1
  while (mytoss(p) == 0){
    counter <- counter + 1
  }
  return(counter)
}

prob <- c(0.04, 0.14, 0.53, 0.85, 0.93)

for (p in prob){
  tic.clearlog()
  for (i in 1:1000){
    tic(i)
    myattempts(p)
    toc(log = TRUE, quiet = TRUE)
```

```
  }
  log.lst <- tic.log(format = FALSE)
  timings <- unlist(lapply(log.lst, function(x) x$toc - x$tic))
  cat('Average time for myattempts when p =', p, 'is', mean(timings), 'sec\n')
  tic.clearlog()
  for (j in 1:1000){
    tic(j)
    1 + rgeom(1, p)
    toc(log = TRUE, quiet = TRUE)
  }
  log.lst <- tic.log(format = FALSE)
  timings <- unlist(lapply(log.lst, function(x) x$toc - x$tic))
  cat('Average time for 1+rgeom(1,p) when p =', p, 'is', mean(timings), 'sec\n')
}
```

```
## Average time for myattempts when p = 0.04 is 0.00011 sec
## Average time for 1+rgeom(1,p) when p = 0.04 is 3e-05 sec
## Average time for myattempts when p = 0.14 is 4e-05 sec
## Average time for 1+rgeom(1,p) when p = 0.14 is 0 sec
## Average time for myattempts when p = 0.53 is 2e-05 sec
## Average time for 1+rgeom(1,p) when p = 0.53 is 0 sec
## Average time for myattempts when p = 0.85 is 4e-05 sec
## Average time for 1+rgeom(1,p) when p = 0.85 is 3e-05 sec
## Average time for myattempts when p = 0.93 is 4e-05 sec
## Average time for 1+rgeom(1,p) when p = 0.93 is 2e-05 sec
```

Note that, from above results, "myattempts" method costs more time than "1+rgeom(1,p)" method, because "myattempts" will always call "mytoss" function until it reaches a successful trial. Also, when the probability p is getting smaller, it would take more trials to succeed. Therefore, time spent will be larger.

Problem 3

Calculate standard deviation of a geometric distribution using formula $\frac{\sqrt{1-p}}{p}$

```
for (p in prob){
  sd_geo <- (sqrt(1-p)/p)
  cat("Standard deviation of geometric distribution (p =", p, ") is", sd_geo, "\n")
}
```

```
## Standard deviation of geometric distribution (p = 0.04 ) is 24.4949
## Standard deviation of geometric distribution (p = 0.14 ) is 6.624013
## Standard deviation of geometric distribution (p = 0.53 ) is 1.29352
## Standard deviation of geometric distribution (p = 0.85 ) is 0.4556451
## Standard deviation of geometric distribution (p = 0.93 ) is 0.2844894
```

Simulate geometric distribution to calculate standard deviation. Since "myattempts(p)" and "1+rgeom(1,p)" are essentially the same thing, I use "1+rgeom(1,p)" in this problem to save running time.

```
for (p in prob){
  data <- replicate(1000, 1 + rgeom(1, p))
  cat("Estimated standard deviation of geometric distribution (p =", p, ") is", sd(data),
      "\n")
}
```

```
## Estimated standard deviation of geometric distribution (p = 0.04 ) is 24.71961
```

```
## Estimated standard deviation of geometric distribution (p = 0.14 ) is 6.893109
## Estimated standard deviation of geometric distribution (p = 0.53 ) is 1.336153
## Estimated standard deviation of geometric distribution (p = 0.85 ) is 0.4559955
## Estimated standard deviation of geometric distribution (p = 0.93 ) is 0.2494358
```

Problem 4

```
replicate(10, mean(rexp(20)))
```

```
##  [1] 0.8390055 0.6184772 1.1805249 0.8820746 1.0090456 0.7792097 0.9045813
##  [8] 1.1795606 0.6453835 1.0854880
```

```
sapply(1:10, function(x) mean(rexp(20)))
```

```
##  [1] 1.1660635 0.9256452 0.7898427 0.7514623 0.7501705 0.8771490 1.4070121
##  [8] 1.1326195 0.9934924 0.9945214
```

Problem 5

Pros

SAS has procedures that could draw maps, as well as graphic template language with map layers. It can deal with extremely large dataset because of its data storage system. SAS is not simply a statistical software, but it is a Business Analytics and Business Intelligence software. It is widely used in finance, communication, energy, government, pharmacy, manufacturing, retailer, etc.

R is a free statistical programming substitute for MATLAB or SAS. It is friendly for beginners. Statements are concise and clear. R has advantages while doing stochastic process analysis. R can create some basic statistical plots. R also has packages that can scrape web.

Cons

SAS's code has its own standard, which is comparatively complicated. SAS is very expensive for personal users and it does not update patches frequently.

R cannot handle huge dataset over hundreds of megabytes. Since it is an open source platform with open codes, developers cannot share packages during development stage.

Problem 6

```
n <- 1000
N <- 1e8
cat("The probability that I will be in this sample is", n/N, '\n')
```

```
## The probability that I will be in this sample is 1e-05
```

```
m <- 2000
cat("The probability that I will not be in any of the samples is", ((N-n)/N)^m, '\n')
```

```
## The probability that I will not be in any of the samples is 0.9801986
```

```r
p <- 0.5
cat(round(log(p, base = (N-n)/N)), "samples must be selected to have a", p,
    "probability of being in at least one sample.")
```

```
## 69314 samples must be selected to have a 0.5 probability of being in at least one sample.
```

Problem 7

```r
prob7 <- c(0.1, 0.3, 0.7)
n <- 4
for (p in prob7){
  data7 <- replicate(n, 1 + rgeom(1, p))
  p_hat <- (sqrt(1 + 8 * mean(data7^2)) - 1)/(2 * mean(data7^2))
  cat("Bias for p =", p, "is", p_hat - p, '\n')
}
```

```
## Bias for p = 0.1 is -0.02548195
## Bias for p = 0.3 is 0.004776867
## Bias for p = 0.7 is -0.1810724
```

Problem 8

```r
n <- 10000
y <- 0
for (i in 1:n){
  x <- runif(1)
  for (i in 2:10){
    x <- runif(1, max = 2 * x)
  }
  y <- y + x
}
cat("The expected value of X_10 is", y/n)
```

```
## The expected value of X_10 is 0.5216509
```