Monte Carlo Simulation Basics in R - Solutions

Exercise A - (4 min)

1. What happens if I don't supply `size` to `sample()`?
2. Does it matter if I set `replace = TRUE` or `FALSE` here?

```
sample(1:6, size = 1)
```

3. Write a function called `rbern()` that uses `sample()` to make a sequence of n iid Bernoulli(p) draws. Its arguments are `n`, the number of draws, and `p`, the probability of success. It should return a vector `n` of zeros and ones. Check your results by making a large number of draws and verifying that the fraction of ones is approximately p.

Solutions

1. It sets `size` to `length(x)`
2. No: since we're only making a single draw from `1:6`, it doesn't matter if we draw with or without replacement.
3. E.g.

```
rbern <- \(n, p) {
  sample(0:1, size = n, replace = TRUE, prob = c(1 - p, p))
}
mean(rbern(1e6, 0.2)) - 0.2
```

```
[1] -0.000149
```

```
mean(rbern(1e6, 0.4)) - 0.4
```

```
[1] 0.00114
```

```
mean(rbern(1e6, 0.6)) - 0.6
```

```
[1] -9.1e-05
```

```
mean(rbern(1e6, 0.8)) - 0.8
```

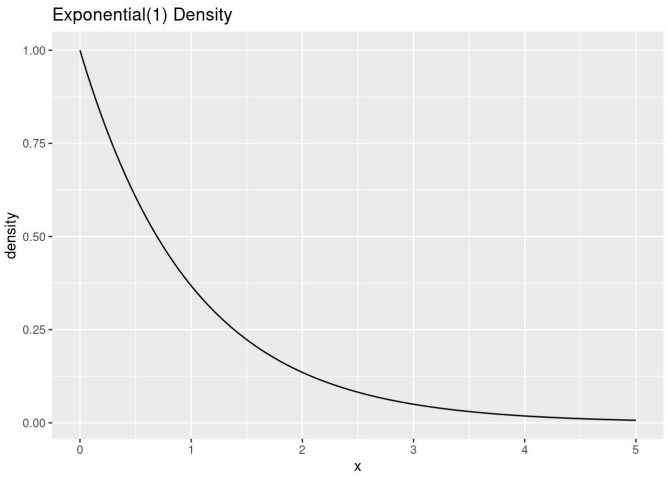
```
[1] 0.000133
```

Exercise B - (10 min)

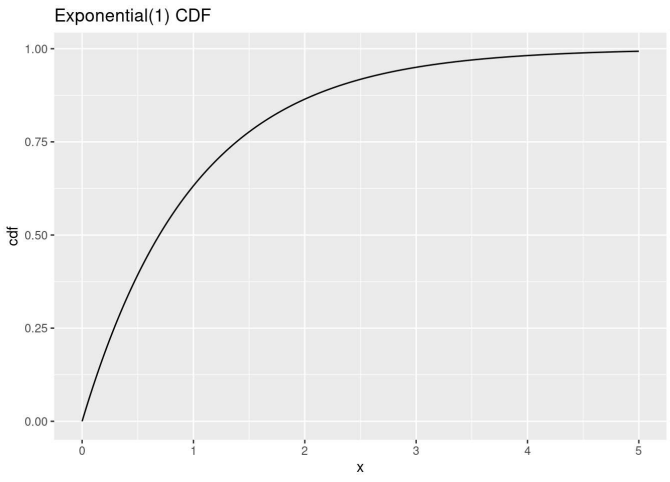
1. Consult `?dexp()` and then carry out the following:
   a. Plot the density and CDF of an Exponential(1) RV.
   b. Plot a histogram of 500 draws from an Exponential(1) distribution.
2. Consult `?dpois()` and then carry out the following:
   a. Plot the mass function and CDF of a Poisson(5) RV.
   b. Plot a barchart of 500 draws from a Poisson(5) distribution.
3. Suppose that we observe a sample of $n = 16$ iid normal RVs with unknown mean μ and known variance $\sigma^2 = 1$.
   a. A confidence interval for μ takes the form $\bar{X} \pm \text{ME}$ where ME is the margin of error. Use `qnorm()` to compute ME for an 89% interval.
   b. Simulate 16 draws from a $N(0, \sigma = 4)$ distribution and construct the interval from part (a). Does it cover the true value of μ?

Solution

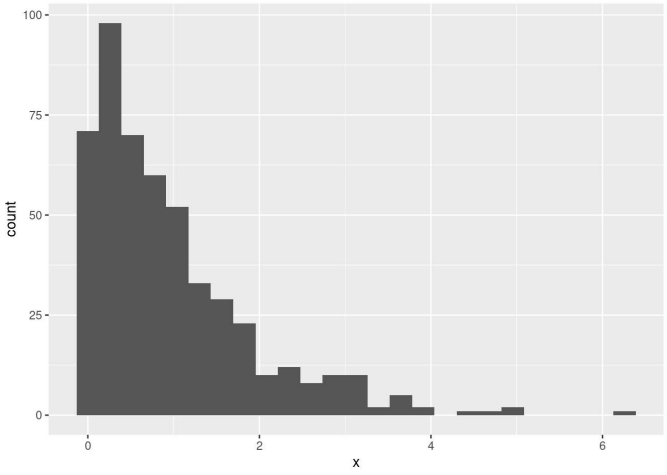
```
library(tidyverse)

# Part 1a
exponential1 <- tibble(x = seq(0, 5, 0.01)) |>
  mutate(density = dexp(x, 1), cdf = pexp(x, 1))

exponential1 |>
  ggplot(aes(x, density)) +
  geom_line() +
  labs(title = 'Exponential(1) Density')
```



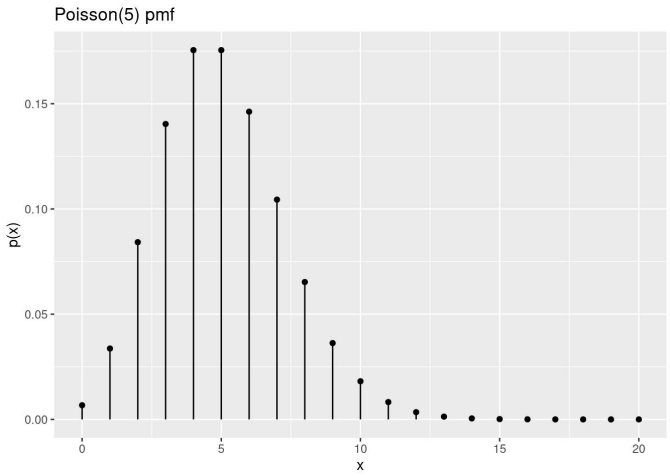

```
exponential1 |>
  ggplot(aes(x, cdf)) +
  geom_line() +
  labs(title = 'Exponential(1) CDF')
```

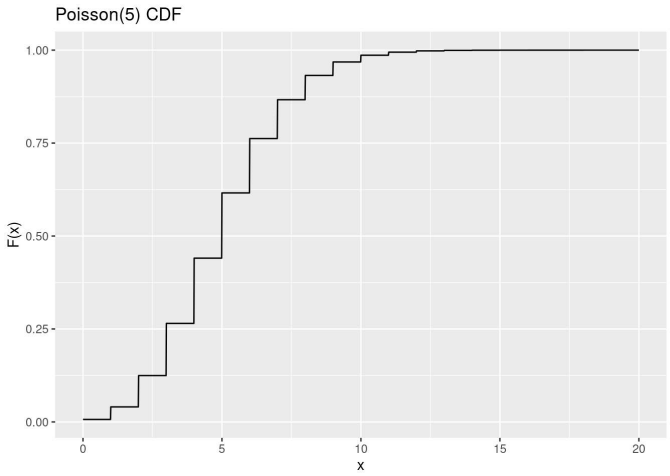



```
# Part 1b
set.seed(690872)
tibble(x = rexp(500, 1)) |>
  ggplot(aes(x)) +
  geom_histogram(bins = 25)
```



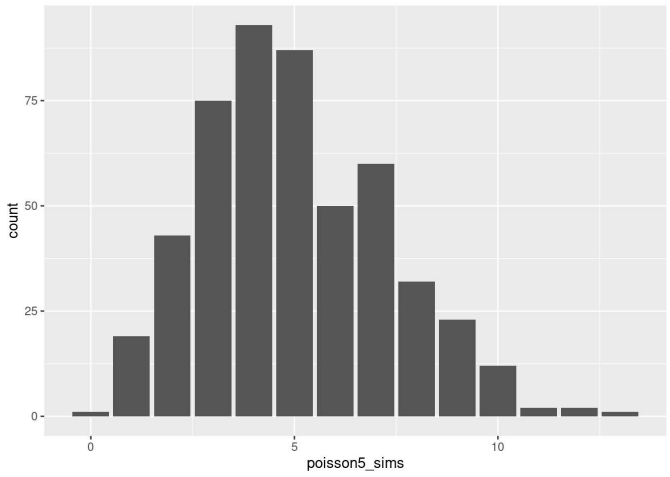

```
# Part 2a
# pmf
tibble(x = 0:20) |>
  mutate(p = dpois(x, 5)) |>
  ggplot(aes(x, p)) +
  geom_point() +
  geom_segment(aes(xend = x, yend = 0)) +
  labs(title = 'Poisson(5) pmf', y = 'p(x)')
```






```
# CDF
tibble(x = seq(0, 20, 0.01)) |>
  mutate(cdf = ppois(x, 5)) |>
  ggplot(aes(x, cdf)) +
  geom_line() +
  labs(title = 'Poisson(5) CDF', y = 'F(x)')
```

```
# Part 2b
poisson5_sims <- rpois(500, 5)
tibble(poisson5_sims) |>
  ggplot(aes(x = poisson5_sims)) +
  geom_bar()
```

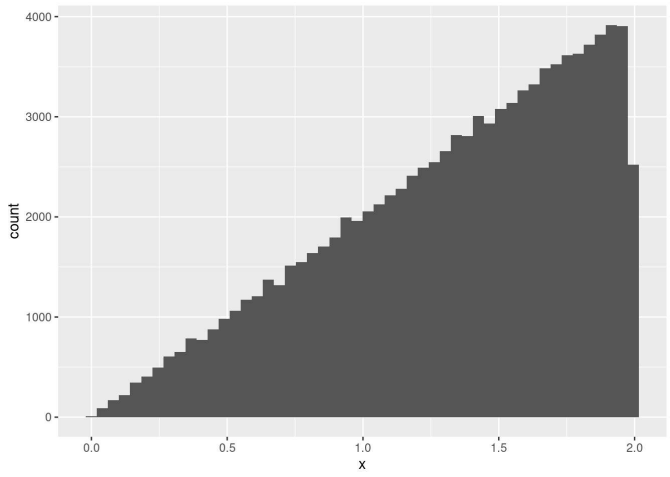


Write code that uses the inverse transformation method to simulate iid draws from this distribution. Test to make sure that your code works correctly.

Solution

Integrating, the CDF of this random variable is $F(x) = x^2/4$. Re-arranging, its quantile function is $F^{-1}(p) = 2\sqrt{p}$.

```
set.seed(987654321)
triangular_sims <- 2 * sqrt(runif(1e5))
tibble(x = triangular_sims) |>
  ggplot(aes(x)) +
  geom_histogram(bins = 50)
```




```
# Part 3a
n <- 16
sigma <- 1
SE <- sigma / sqrt(n)
alpha <- (1 - 0.89)
ME <- qnorm(1 - alpha / 2) * SE
ME # Just under 0.4
```

```
[1] 0.3995483
```

```
# Part 3b
xbar <- rnorm(n, mean = 0, sd = sigma) |>
  mean()
CI <- xbar + c(-1, 1) * ME
CI
```

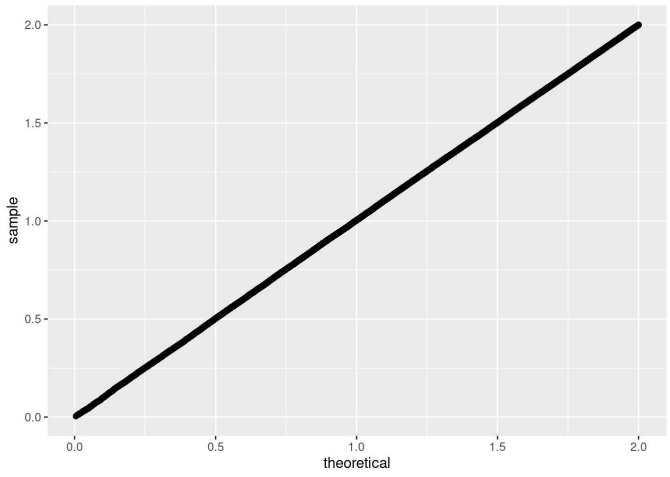
```
[1] -0.03107317  0.76802340
```

```
tibble(x = triangular_sims) |>
  ggplot(aes(sample = x)) +
  geom_qq(distribution = \(p) 2 * sqrt(p))
```

Exercise C - (10 min)

Let X be a random variable with the following density function

$$f(x) = \begin{cases} x/2, & x \in [0, 2] \\ 0, & \text{otherwise.} \end{cases}$$



Exercise D - (15 min)

This problem was initially posed by the famous 17th century gambler Antoine Gombaud, more commonly known as the Chevalier de Méré. Fermat and Pascal discussed its solution in their legendary correspondance that began the study of probability as we know it. Here's the Chevalier's question:

> Which is more likely: (A) getting at least one six when rolling a single die four times or (B) getting at least one pair of sixes when rolling a pair of dice twenty-four times?

Answer the Chevalier's question using Monte Carlo Simulation. Assume that all dice are fair and six-sided.

Solution

```
experimentA <- function(){
  rolls <- sample(1:6, size = 4, replace = TRUE)
  condition <- sum(rolls == 6) > 0
  return(condition)
}

experimentB <- function(){
  die1 <- sample(1:6, size = 24, replace = TRUE)
  die2 <- sample(1:6, size = 24, replace = TRUE)
  condition <- sum((die1 == die2) & (die1 == 6)) > 0
  return(condition)
}
```

```
nreps <- 1e5

simsA <- map_dbl(1:nreps, \(i) experimentA())
mean(simsA)
```

```
[1] 0.51659
```

```
simsB <- map_dbl(1:nreps, \(i) experimentB())
mean(simsB)
```

```
[1] 0.49192
```

It appears that A is *slightly* more likely than B. We might wonder, however, if this is a real difference, or merely a chance fluctuation that arose in our simulation. A simple way to assess this is by computing a standard error for the difference of estimated probabilities. Let p_A be the probability of A and p_B be the probability of B. We have estimates $\widehat{p}_A$ and $\widehat{p}_B$, each obtained from an independent sample of `nreps` observations. Hence, we have

$$\text{SE}(\widehat{p}_A - \widehat{p}_B) = \sqrt{\frac{p_A(1-p_A)}{\text{nreps}} + \frac{p_B(1-p_B)}{\text{nreps}}}.$$

Unfortunately the standard error depends on p_A and p_B which we don't know! One option would be to substitute our estimates of these values to approximate the standard error. Another option is to compute the *worst case* standard error, by plugging in the values of p_A and p_B that make the above function as large as possible. The "actual" standard error will necessarily be no larger than this worst case value. A bit of calculus shows that setting $p_A = p_B = 1/2$ maximizes $\text{SE}(\widehat{p}_A - \widehat{p}_B)$. Hence, the worst-case standard error is

```
SE_worst <- sqrt(0.5 * (1 - 0.5) / nreps + 0.5 * (1 - 0.5) / nreps)
SE_worst
```

```
[1] 0.002236068
```

This is very small compared to our estimated difference. For example, we can construct a $> 99.7\%$ confidence interval for $(p_A - p_B)$ as follows:

```
(mean(simsA) - mean(simsB)) + c(-1, 1) * 3 * SE_worst
```

```
[1] 0.0179618 0.0313782
```

This means we can be extremely confidence that p_A is indeed slightly larger than p_B. If the confidence interval had come out too wide for our liking, we could simply go back and choose a larger value of `nreps`. Isn't simulation great?!

Exercise E - (10 min)

Find an alternative, faster way to carry out the "sum of two fair dice" simulation from above. Hint, think about how you can *avoid* writing a function to carry out the simulation once and do the whole thing in a single step. Time your alternative approach to see how much faster it is.

Solution

Use vectorized operations

```
dice_sum <- \() {
# Roll a pair of fair, six-sided dice and return their sum
  die1 <- sample(1:6, 1)
  die2 <- sample(1:6, 1)
  die1 + die2
}

version1 <- function(nreps) {
  map_dbl(1:nreps, \(i) dice_sum())
}

version2 <- function(nreps) {
  die1 <- sample(1:6, nreps, replace = TRUE)
  die2 <- sample(1:6, nreps, replace = TRUE)
  die1 + die2
}

set.seed(1234)
system.time(
  sims1 <- version1(1e5)
)
```

```
   user  system elapsed
  0.650   0.001   0.651
```

```
system.time(
  sims2 <- version2(1e5)
)
```

```
   user  system elapsed
  0.005   0.000   0.005
```