STA 3100 Programming with Data: Assignment 050

(60 points)

The Coupon Collector's Problem

Beginning in the late 19th century, tobacco companies, such as the British manufacturer John Player and Son's, often included collector's cards in packets of cigarettes. Each packet contained one card, and a typical series would consist of 25 to 50 related cards. This gives rise to a version of the famous coupon collector's problem: how many packets does a consumer need to to buy in order to obtain a complete set of the cards/coupons?

Of course the number of purchases required is random, so a better question is:

> What is the probability distribution of the number of purchases required to get a complete set of coupons?

To answer this question, we will make a few (reasonable) simplifying assumptions. Let n represent the number of cards in a complete set.

a. Assume that each purchase is a random draw from the set of available cards.

b. Assume that the manufacturer produces the same number of each card in the collection.

c. Assume that the total number of cards manufactured is large enough that our purchases do not appreciably change the population of cards remaining be purchased.

The idea is that we are sampling randomly from an extremely large population in which the cards in the series appear in equal proportions. Because the sample size is very small relative to the size of the population, we can pretend that we are sampling *with replacement* from the series of cards.

There is an obvious direct way to simulate this experiment: draw one card at a time until we have at least one card of each type. Here is a function that uses this approach to simulate the number of purchases required to obtain one complete set:

```
coupon_slow <- function(n) {
    coupons <- rep(0L, n)
    while (any(coupons == 0L)) {
        new_coupon <- sample.int(n, 1)
        coupons[new_coupon] <- coupons[new_coupon] + 1
    }
    sum(coupons)
}
```

However, we can speed things up by thinking more carefully about the problem. Let:

- X_1 be the number of draws needed to get our first card from the collection;

- X_2 be the number of additional draws needed to get a card different from the one we already have;
- X_3 be the number of additional draws needed to get a card different from the first two;
- etc.

Of course $X_1 \equiv 1$. Then, on our next purchase the probability that we get a card different from the first one is just $\frac{n-1}{n}$, and we will continue to buy packets with this same probability of "success" until we do get a card different from the first one, with each purchase being independent of the last (the assumption of a practically infinite population of packets plays a role here). This means that X_2 is a *geometrically distributed*[1] random variable, with $p_2 = \frac{n-1}{n}$.

Then, having obtained two distinct cards from the collection, we begin drawing to get a card different from the first two. At each attempt, the probability that we get a card different from the first two is $\frac{n-2}{n}$, and X_3, the number of draws required to achieve this goal is geometrically distributed with $p_3 = \frac{n-2}{n}$. Note also that X_3 is statistically independent of X_1 and X_2: the number of packets that we had to purchase to get 2 different cards has no influence on the distribution of the number of additonal packets we must buy to get our 3rd distinct card (again, the assumption of a practically infinite population of packets justifies this statement).

Continuing in this way, we see that the total number of purchase required to get a complete set of n distinct cards is

$$S = X_1 + X_2 + \cdots + X_n,$$

where $X_1, X_2, \ldots, X_n$ are independent and each X_k is geometrically distributed with

$$p_k = \frac{n-(k-1)}{n} = 1 - \frac{k-1}{n}.$$

This is useful in more than one way. For example, it is well known that a geometric random variable has expected value (or mean) $\frac{1}{p}$, and since the expected value of a sum of random variables is the sum of their expectations, we can easily calculate the expected value of S. Similarly, the variance of a geometric random variable is $\frac{1-p}{p^2}$, and because the variance of a sum of *independent* random variables is equal to the sum of their variances, we can also easily calculate the variance and hence the standard deviation of S.[2]

This formulation also allows us to simulate the number of purchases required to obtain a complete set of cards in a much more efficient way.

```
coupon_fast <- function(n) {
    p <- 1 - (0:(n-1))/n
    x <- rgeom(n, p)  ## R's geometric distriubtion counts only the failures before the first success
    n + sum(x)        ## so add n * 1 = n to get the correct total.
}
```

Exercises

1. (10 pts) Write a function named `coupon_mean_sd` which computes the (theoretical/population) mean and standard deviation of S. Your function should return the mean and standard deviation as a named vector or a named list. Use your function to compute the mean and standard deviation when $n = 25$, $n = 50$, and $n = 100$.

[1] We take the geometric distribution to represent distribution of the number of *trials* required to obtain the first success in a sequence of independent trials where the probability of a success in each trial is p. Some authors count the number of *failures* before the first sucess, so their geometric random variable is exactly one less than ours.

[2] In principal, we could also compute the exact distribution of S, but this would be beyond the scope of this class.

```r
coupon_mean_sd <- function(n)
{
  p <- 1 - (0:(n-1))/n
  mean <- sum(1/p)
  sd <- sqrt(sum((1/p)/(p^2)))
  mean_sd <- c("Mean" = mean, "SD" = sd)
  return(mean_sd)
}
coupon_mean_sd(25)
```

```
##      Mean        SD
##  95.39895 137.00412
```

```r
coupon_mean_sd(50)
```

```
##     Mean       SD
## 224.9603 387.5985
```

```r
coupon_mean_sd(100)
```

```
##      Mean        SD
##  518.7378 1096.3610
```

2. (10 pts) Use `system.time()` to compare the *elapsed* times required for `coupon_slow()` and `coupon_fast()` to perform 1,000 replications of the coupon collectors problem for $n = 25$, $n = 50$, and $n = 100$. How many times slower is the slow function than the fast version? Does the speed ratio depend on n?

```r
# tried to use replicate, rep, and lapply, but all were too fast and I kept getting 0,0,0 as times

replications_slow <- function(n){
  for (i in 1:1000){
    coupon_slow(n)
  }
}

replications_fast <- function(n){
  for (i in 1:1000){
    coupon_fast(n)
  }
}

system.time(replications_slow(25))
```

```
##    user  system elapsed
##    0.33    0.06    0.39
```

```r
system.time(replications_fast(25))
```

```
##    user  system elapsed
##    0.00    0.02    0.01
```

```
system.time(replications_slow(50))
```

```
##    user  system elapsed
##    0.78    0.03    0.81
```

```
system.time(replications_fast(50))
```

```
##    user  system elapsed
##       0       0       0
```

```
system.time(replications_slow(100))
```

```
##    user  system elapsed
##    1.95    0.00    1.97
```

```
system.time(replications_fast(100))
```

```
##    user  system elapsed
##    0.02    0.00    0.02
```

The slow version of the function is a lot slower than the fast version. When n = 25, the slow version is 0.5 seconds slower. When n = 50, the slow version is 0.95 seconds slower. Finally when n = 100, the slow version is 2.28 seconds slower.

The speed of the functions do depend on n. For example the slow function increased in time elapsed from 0.5 to 2.30 from when n = 25 to when n = 100. Similarly, the fast function increased from being marked as 0 seconds when n = 25 to taking 0.02 seconds when n = 50 and 100. Granted a tiny increase in comparison to the slow function.

3. Estimating the distribution of S via simulation.

    a. (5 pts) For $n = 50$, use `coupon_fast()` to compute 10,000 simulated values of S.

    b. (5 pts) Compare the sample mean and standard deviation of your 10,000 S values to the theoretical values. Do they agree reasonably well? (Note: a large discrepancy would suggest that you may have made an error either in computing the theoretical mean and standard deviation or in your simulation code.)

    c. (5 pts) Use `hist()` to plot the estimated distribution of S. (Setting `nclass` to anything from 25 to 50 or so seems to work pretty well here. Use your judgement.)

    d. (5 pts) Estimate the 10th, 50th, and 90th percentiles of S.

4. *What if we were interested in the number of number of trials required to get m complete sets?* Here is an adaptation of our earlier `coupon_fast()` function for this problem. Because of the additional bookkeeping required, its speed advantage over the slow version is much less pronounced. (Of course it is possible that there is a better way to do this that I haven't thought of.)

```
mcoupon_fast <- function(n, m) {
    coupons <- rep(0L, n)  # Will count each one until we have m of that coupon, then stop
    incomplete <- (coupons < m)
    S <- 0
    while (any(incomplete)) {
```

```
        k <- sum(incomplete)    # Number of coupons with fewer than m accumulated so far
        x <- rgeom(1, k/n) + 1  # Success is getting any one of the coupons we still need
        S <- S + x              # Update number of purchases
        y <- sample.int(k, 1)   # Which of the not-yet-completed coupons did we draw
        coupons[incomplete][y] <- coupons[incomplete][y] + 1   # Increment incomplete coupon counts
        incomplete <- (coupons < m)
    }
    S
}
```

a. (5 pts) Create a new function `mcoupon_slow()` by adding a second argument `m` to `coupon_slow()` and modifying the function so that it simulates this more general version of the coupon collecting problem.

b. (10 pts) Using `mcoupon_fast()`, generate 1,000 simulated values of the number of packets required to obtain m complete sets of a series of $n = 25$ cards for $m = 1, 2, \ldots, 8$. Calculate the sample mean $\bar{S}_m$ for $m = 1, 2, \ldots, 8$ and plot the points $(m, \bar{S}_m)$. Judging from your plot, it is plausible that the true mean number of packets required depends linearly on m (with $n = 25$ fixed)?

c. (5 pts) One might guess that on average it takes twice as many packets to get two complete sets as it does to get one. Do your results suggest that this is true? What would you estimate the ratio to be base on your simulation results?