Generating Discrete Random Variables

4.1 The Inverse Transform Method

Suppose we want to generate the value of a discrete random variable X having probability mass function

$$P\{X = x_j\} = p_j, \quad j = 0, 1, \cdots, \sum_j p_j = 1$$

To accomplish this, we generate a random number U–that is, U is uniformly distributed over (0, 1)–and set

$$X = \begin{cases} x_0 & \text{If } U < p_0 \\ x_1 & \text{If } p_0 \le U < p_0 + p_1 \\ \vdots & \\ x_j & \text{If } \sum_{i=0}^{j-1} p_i \le U < \sum_{i=0}^{j} p_i \\ \vdots & \end{cases}$$

Since, for $0 < a < b < 1$, $p\{a \le U < b\} = b - a$, we have that

$$p\{X = x_j\} = p\left\{\sum_{i=0}^{j-1} p_i \le U < \sum_{i=0}^{j} p_i\right\} = p_j$$

and so X has desired distribution.

Simulation. DOI: http://dx.doi.org/10.1016/B978-0-12-415825-2.00004-8

Remarks

1. The preceding can be written algorithmically as

$$\begin{aligned}&\text{Generate a random number } U\\ &\text{If } \quad U < p_0 \text{ set } X = x_0 \text{ and } \quad \text{stop}\\ &\text{If } \quad U < p_0 + p_1 \text{ set } X = x_1 \text{and } \quad \text{stop}\\ &\text{If } \quad U < p_0 + p_1 + p_2 \text{ set } X = x_2 \text{ and } \quad \text{stop}\\ &\vdots\end{aligned}$$

2. If the $x_i, i \geq 0$, are ordered so that $x_0 < x_1 < x_2 < \cdots$ and if we let F denote the distribution function of X, then $F(x_k) = \sum_{i=0}^{k} p_i$ and so

$$X \quad \text{will equal } x_j \quad \text{if} \quad F(x_{j-1}) \leq U < F(x_j)$$

In other words, after generating a random number U we determine the value of X by finding the interval $[F(x_{j-1}), F(x_j)$ in which U lies [or, equivalently, by finding the inverse of $F(U)$]. It is for this reason that the above is called the discrete inverse transform method for generating X.

The amount of time it takes to generate a discrete random variable by the above method is proportional to the number of intervals one must search. For this reason it is sometimes worthwhile to consider the possible values x_j of X in decreasing order of the p_j.

Example 4a If we wanted to simulate a random variable X such that

$$p_1 = 0.20, \quad p_2 = 0.15, \quad p_3 = 0.25, \quad p_4 = 0.40 \quad \text{where } p_j = P\{X = j\}$$

then we could generate U and do the following:

$$\begin{aligned}&\text{If } \quad U < 0.20 \text{ set } X = 1 \text{ and stop}\\ &\text{If } \quad U < 0.35 \text{ set } X = 2 \text{ and stop}\\ &\text{If } \quad U < 0.60 \text{ set } X = 3 \text{ and top}\\ &\text{Otherwise set } X = 4\end{aligned}$$

However, a more efficient procedure is the following:

$$\begin{aligned}&\text{If } \quad U < 0.40 \text{ set } X = 4 \text{ and stop}\\ &\text{If } \quad U < 0.65 \text{ set } X = 3 \text{ and stop}\\ &\text{If } \quad U < 0.85 \text{ set } X = 1 \text{ and stop}\\ &\text{Otherwise set } X = 2\end{aligned}$$

□

One case where it is not necessary to search for the appropriate interval in which the random number lies is when the desired random variable is the

discrete uniform random variable. That is, suppose we want to generate the value of X which is equally likely to take on any of the values $1, \ldots, n$. That is, $P\{X = j\} = 1/n, j = 1, \ldots, n$. Using the preceding results it follows that we can accomplish this by generating U and then setting

$$X = j \quad \text{if} \quad \frac{j-1}{n} \leq U < \frac{j}{n}$$

Therefore, X will equal j if $j - 1 \leq nU < j$; or, in other words,

$$X = \text{Int}(nU) + 1$$

where Int (x)—sometimes written as $[x]$—is the integer part of x (i.e., the largest integer less than or equal to x).

Discrete uniform random variables are quite important in simulation, as is indicated in the following two examples.

Example 4b Generating a Random Permutation Suppose we are interested in generating a permutation of the numbers $1, 2, \ldots, n$ which is such that all $n!$ possible orderings are equally likely. The following algorithm will accomplish this by first choosing one of the numbers $1, \ldots, n$ at random and then putting that number in position n; it then chooses at random one of the remaining $n - 1$ numbers and puts that number in position $n - 1$; it then chooses at random one of the remaining $n - 2$ numbers and puts it in position $n - 2$; and so on (where choosing a number at random means that each of the remaining numbers is equally likely to be chosen). However, so that we do not have to consider exactly which of the numbers remain to be positioned, it is convenient and efficient to keep the numbers in an ordered list and then randomly choose the position of the number rather than the number itself. That is, starting with any initial ordering $P_1, P_2, \ldots, P_n$ we pick one of the positions $1, \ldots, n$ at random and then interchange the number in that position with the one in position n. Now we randomly choose one of the positions $1, \ldots, n-1$ and interchange the number in this position with the one in position $n - 1$, and so on.

Recalling that $\text{Int}(kU) + 1$ will be equally likely to take on any of the values $1, 2, \ldots, k$, we see that the above algorithm for generating a random permutation can be written as follows:

STEP 1: Let $P_1, P_2, \ldots, P_n$ be any permutation of $1, 2, \ldots, n$ (e.g., we can choose $P_j = j, j = 1, \ldots, n$).
STEP 2: Set $k = n$.
STEP 3: Generate a random number U and let $I = \text{Int}(kU) + 1$.
STEP 4: Interchange the values of P_I and P_k.
STEP 5: Let $k = k - 1$ and if $k > 1$ go to Step 3.
STEP 6: $P_1, \ldots, P_n$ is the desired random permutation.

For instance, suppose $n = 4$ and the initial permutation is 1, 2, 3, 4. If the first value of I (which is equally likely to be either 1, 2, 3, or 4) is $I = 3$, then the

elements in positions 3 and 4 are interchanged and so the new permutation is 1, 2, 4, 3. If the next value of I is $I = 2$, then the elements in positions 2 and 3 are interchanged and so the new permutation is 1, 4, 2, 3. If the final value of I is $I = 2$, then the final permutation is 1, 4, 2, 3, and this is the value of the random permutation. □

One very important property of the preceding algorithm is that it can also be used to generate a random subset, say of size r, of the integers $1, \ldots, n$. Namely, just follow the algorithm until the positions $n, n-1, \ldots, n-r+1$ are filled. The elements in these positions constitute the random subset. (In doing this we can always suppose that $r \leq n/2$; for if $r > n/2$ then we could choose a random subset of size $n-r$ and let the elements not in this subset be the random subset of size r.)

It should be noted that the ability to generate a random subset is particularly important in medical trials. For instance, suppose that a medical center is planning to test a new drug designed to reduce its user's blood cholesterol level. To test its effectiveness, the medical center has recruited 1000 volunteers to be subjects in the test. To take into account the possibility that the subjects' blood cholesterol levels may be affected by factors external to the test (such as changing weather conditions), it has been decided to split the volunteers into two groups of size 500—a *treatment* group that will be given the drug and a *control* that will be given a placebo. Both the volunteers and the administrators of the drug will not be told who is in each group (such a test is called *double-blind*). It remains to determine which of the volunteers should be chosen to constitute the treatment group. Clearly, one would want the treatment group and the control group to be as similar as possible in all respects with the exception that members in the first group are to receive the drug while those in the other group receive a placebo, for then it would be possible to conclude that any difference in response between the groups is indeed due to the drug. There is general agreement that the best way to accomplish this is to choose the 500 volunteers to be in the treatment group in a completely random fashion. That is, the choice should be made so that each of the $\binom{1000}{500}$ subsets of 500 volunteers is equally likely to constitute the set of volunteers.

Remarks Another way to generate a random permutation is to generate n random numbers $U_1, \ldots, U_n$, order them, and then use the indices of the successive values as the random permutation. For instance, if $n = 4$, and $U_1 = 0.4, U_2 = 0.1, U_3 = 0.8, U_4 = 0.7$, then, because $U_2 < U_1 < U_4 < U_3$, the random permutation is 2, 1, 4, 3. The difficulty with this approach, however, is that ordering the random numbers typically requires on the order of $n \log(n)$ comparisons. □

Example 4c Calculating Averages Suppose we want to approximate $\overline{a} = \sum_{i=1}^{n} a(i)/n$, where n is large and the values $a(i), i = 1, \ldots, n$, are complicated and not easily calculated. One way to accomplish this is to note that if X is a discrete uniform random variable over the integers $1, \ldots, n$, then the random variable $a(X)$ has a mean given by

$$E[a(X)] = \sum_{i=1}^{n} a(i)P\{X = i\} = \sum_{i=1}^{n} \frac{a(i)}{n} = \overline{a}$$

Hence, if we generate k discrete uniform random variables $X_i, i = 1, \ldots, k$—by generating k random numbers U_i and setting $X_i = \text{Int}(nU_i) + 1$—then each of the k random variables $a(X_i)$ will have mean $\overline{a}$, and so by the strong law of large numbers it follows that when k is large (though much smaller than n) the average of these values should approximately equal $\overline{a}$. Hence, we can approximate $\overline{a}$ by using

$$\overline{a} \approx \sum_{i=1}^{k} \frac{a(X_i)}{k}$$ □

Another random variable that can be generated without needing to search for the relevant interval in which the random number falls is the geometric.

Example 4d Recall that X is said to be a geometric random variable with parameter p if

$$P\{X = i\} = pq^{i-1}, \quad i \geq 1, \quad \text{where } q = 1 - p$$

X can be thought of as representing the time of the first success when independent trials, each of which is a success with probability p, are performed. Since

$$\begin{aligned} \sum_{i=1}^{j-1} P\{X = i\} &= 1 - P\{X > j - 1\} \\ &= 1 - P\{\text{first } j - 1 \text{ trials are all failures }\} \\ &= 1 - q^{j-1}, \quad j \geq 1 \end{aligned}$$

we can generate the value of X by generating a random number U and setting X equal to that value j for which

$$1 - q^{j-1} \leq U < 1 - q^{j}$$

or, equivalently, for which

$$q^{j} < 1 - U \leq q^{j-1}$$

That is, we can define X by

$$X = \text{Min}\{j: q^{j} < 1 - U\}$$

Hence, using the fact that the logarithm is a monotone function, and so $a < b$ is equivalent to $\log(a) < \log(b)$, we obtain that X can be expressed as

$$\begin{aligned} X &= \text{Min}\{j\colon j\log(q) < \log(1-U)\} \\ &= \text{Min}\left\{j\colon j > \frac{\log(1-U)}{\log(q)}\right\} \end{aligned}$$

where the last inequality changed sign because $\log(q)$ is negative for $0 < q < 1$. Hence, using Int ( ) notation we can express X as

$$X = \text{Int}\left(\frac{\log(1-U)}{\log(q)}\right) + 1$$

Finally, by noting that $1 - U$ is also uniformly distributed on (0, 1), it follows that

$$X \equiv \text{Int}\left(\frac{\log(U)}{\log(q)}\right) + 1$$

is also geometric with parameter p. □

Example 4e Generating a Sequence of Independent Bernoulli Random Variables Suppose that you want to generate n independent and identically distributed Bernoulli random variables $X_1, \ldots, X_n$ with parameter p. While this is easily accomplished by generating n random numbers $U_1, \ldots, U_n$ and then setting

$$X_i = \begin{cases} 1, & \text{if} \quad U_i \le p \\ 0, & \text{if} \quad U_i > p \end{cases}$$

we will now develop a more efficient approach. To do so, imagine these random variables represent the result of sequential trials, with trial i being a success if $X_i = 1$ or a failure otherwise. To generate these trials when $p \le 1/2$, use the result of the Example 4d to generate the geometric random variable N, equal to the trial number of the first success when all trials have success probability p. Suppose the simulated value of N is $N = j$. If $j > n$, set $X_i = 0, i = 1, \ldots, n$; if $j \le n$, set $X_1 = \ldots = X_{j-1} = 0, X_j = 1$; and, if $j < n$, repeat the preceding operation to obtain the values of the remaining $n - j$ Bernoulli random variables. (When $p > 1/2$, because we want to simultaneously generate as many Bernoulli variables as possible, we should generate the trial number of the first failure rather than that of the first success.)

The preceding idea can also be applied when the X_i are independent but not identically distributed Bernoulli random variables. For each $i = 1, \ldots, n$, let u_i be the least likely of the two possible values of X_i. That is, $u_i = 1$ if $P\{X_i = 1\} \le 1/2$, and $u_i = 0$ otherwise. Also, let $p_i = P\{X_i = u_i\}$ and let $q_i = 1 - p_i$. We will simulate the sequence of Bernoullis by first generating the value of X, where for $j = 1, \ldots, n$, X will equal j when trial j is the first trial that results in an unlikely

value, and X will equal $n + 1$ if none of the n trials results in its unlikely value. To generate X, let $q_{n+1} = 0$ and note that

$$P\{X > j\} = \prod_{i=1}^{j} q_i, \; j = 1, \ldots, n + 1$$

Thus,

$$P\{X \leq j\} = 1 - \prod_{i=1}^{j} q_i, \; j = 1, \ldots, n + 1$$

Consequently, we can simulate X by generating a random number, U, and then setting

$$X = \min \left\{ j: U \leq 1 - \prod_{i=1}^{j} q_i \right\}$$

If $X = n + 1$, the simulated sequence of Bernoulli random variables is $X_i = 1 - u_i, i = 1, \ldots, n$. If $X = j, j \leq n$, set $X_i = 1 - u_i, i = 1, \ldots, j - 1, X_j = u_j$; if $j < n$ then generate the remaining values $X_{j+1}, \ldots, X_n$ in a similar fashion.

Remark on Reusing Random Numbers Although the procedure just given for generating the results of n independent trials is more efficient than generating a uniform random variable for each trial, in theory one could use a single random number to generate all n trial results. To do so, start by generating a random U and letting

$$X_1 = \begin{cases} 1, & \text{if } U \leq p_1 \\ 0, & \text{if } U > p_1 \end{cases}$$

Now, use that the conditional distribution of U given that $U \leq p$ is the uniform distribution on $(0, p)$. Consequently, given that $U \leq p_1$, the ratio $\frac{U}{P_1}$ is uniform on (0, 1). Similarly, using that the conditional distribution of U given that $U > p$ is the uniform distribution on $(p, 1)$, it follows that conditional on $U > p_1$ the ratio $\frac{U-p_1}{1-p_1}$ is uniform on (0,1). Thus, we can in theory use a single random number U to generate the results of the n trials as follows:

1. I = 1
2. Generate U
3. If $U \leq p_I$ set $X_I = 1$, otherwise set $X_I = 0$
4. If $I = n$ stop
5. If $U \leq p_I$ set $U = \frac{U}{p_I}$, otherwise set $U = \frac{U-p_I}{1-p_I}$
6. $I = I + 1$
7. Go to Line 3.

There is, however, a practicle problem with reusing a single random number; namely, that computers only specify random numbers up to a certain number of decimal places, and round off errors can result in the transformed variables becoming less uniform after awhile. For instance, suppose in the preceding that all $p_i = .5$. Then U is transformed either to $2U$ if $U \leq .5$, or $2U - 1$ if $U > .5$. Consequently, if the last digit of U is 0 then it will remain 0 in the next transformation. Also, if the next to last digit ever becomes 5 then it will be transformed to 0 in the next iteration, and so the last 2 digits will always be 0 from then on, and so on. Thus, if one is not careful all the random numbers could end up equal to 1 or 0 after a large number of iterations. (One possible solution might be to use $2U - .999\ldots9$ rather than $2U - 1$.)

4.2 Generating a Poisson Random Variable

The random variable X is Poisson with mean λ if

$$p_i = P\{X = i\} = e^{-\lambda}\frac{\lambda^i}{i!} \quad i = 0, 1, \ldots$$

The key to using the inverse transform method to generate such a random variable is the following identity (proved in Section 2.8 of Chapter 2):

$$p_{i+1} = \frac{\lambda}{i+1}p_i, \quad i \geq 0 \tag{4.1}$$

Upon using the above recursion to compute the Poisson probabilities as they become needed, the inverse transform algorithm for generating a Poisson random variable with mean λ can be expressed as follows. (The quantity i refers to the value presently under consideration; $p = p_i$ is the probability that X equals i, and $F = F(i)$ is the probability that X is less than or equal to i.)

STEP 1: Generate a random number U.
STEP 2: $i = 0$, $p = e^{-\lambda}$, $F = p$.
STEP 3: If $U < F$, set $X = i$ and stop.
STEP 4: $p = \lambda p/(i + 1)$, $F = F + p$, $i = i + 1$.
STEP 5: Go to Step 3.

(In the above it should be noted that when we write, for example, $i = i + 1$, we do not mean that i is equal to $i + 1$ but rather that the value of i should be increased by 1.) To see that the above algorithm does indeed generate a Poisson random variable with mean λ, note that it first generates a random number U and then checks whether or not $U < e^{-\lambda} = p_0$. If so, it sets $X = 0$. If not, then it computes (in Step 4) p_1 by using the recursion (4.1). It now checks whether $U < p_0 + p_1$ (where the right-hand side is the new value of F), and if so it sets $X = 1$, and so on.

The above algorithm successively checks whether the Poisson value is 0, then whether it is 1, then 2, and so on. Thus, the number of comparisons needed will be 1 greater than the generated value of the Poisson. Hence, on average, the above will need to make $1 + \lambda$ searches. Whereas this is fine when λ is small, it can be greatly improved upon when λ is large. Indeed, since a Poisson random variable with mean λ is most likely to take on one of the two integral values closest to λ, a more efficient algorithm would first check one of these values, rather than starting at 0 and working upward. For instance, let $I = \text{Int}(\lambda)$ and use Equation (4.1) to recursively determine $F(I)$. Now generate a Poisson random variable X with mean λ by generating a random number U, noting whether or not $X \leq I$ by seeing whether or not $U \leq F(I)$. Then search downward starting from I in the case where $X \leq I$ and upward starting from $I + 1$ otherwise.

The number of searches needed by this algorithm is roughly 1 morethan the absolute difference between the random variable X and its mean λ. Since for λ large a Poisson is (by the central limit theorem) approximately normal with mean and variance both equal to λ, it follows that[1]

$$\begin{aligned}
&\text{Average number of searches } \simeq 1 + E[|X - \lambda|] \qquad \text{where } X \sim N(\lambda, \lambda)^* \\
&= 1 + \sqrt{\lambda} E\left[\frac{|X - \lambda|}{\sqrt{\lambda}}\right] \\
&= 1 + \sqrt{\lambda} E\,[|Z|] \qquad \text{where } Z \sim N(0, 1) \\
&= 1 + 0.798\sqrt{\lambda} \qquad (\text{ see Exercise 11})
\end{aligned}$$

That is, using Algorithm 4-1, the average number of searches grows with the square root of λ rather than with λ as λ becomes larger and larger.

4.3 Generating Binomial Random Variables

Suppose we want to generate the value of a binomial (n, p) random variable X—that is, X is such that

$$P\{X = i\} = \frac{n!}{i!(n-i)!} p^i (1-p)^{n-i}, \quad i = 0, 1, \ldots, n$$

To do so, we employ the inverse transform method by making use of the recursive identity

$$P\{X = i + 1\} = \frac{n - i}{i + 1} \frac{p}{1 - p} P\{X = i\}$$

[1] We use the notation $X \sim F$ to mean that X has distribution function F. The symbol $N(\mu, \sigma^2)$ stands for a normal distribution with mean μ and variance σ^2.

With i denoting the value currently under consideration, pr $= P\{X = i\}$ the probability that X is equal to i, and $F = F(i)$ the probability that X is less than or equal to i, the algorithm can be expressed as follows:

Inverse Transform Algorithm for Generating a Binomial (n, p) Random Variable

STEP 1: Generate a random number U.
STEP 2: $c = p/(1-p)$, $i = 0$, pr $= (1-p)^n$, $F =$ pr.
STEP 3: If $U < F$, set $X = i$ and stop.
STEP 4: pr $= [c(n-i)/(i+1)]$ pr, $F = F +$ pr, $i = i + 1$.
STEP 5: Go to Step 3.

The preceding algorithm first checks whether $X = 0$, then whether $X = 1$, and so on. Hence, the number of searches it makes is 1 more than the value of X. Therefore, on average, it will take $1 + np$ searches to generate X. Since a binomial (n, p) random variable represents the number of successes in n independent trials when each is a success with probability p, it follows that such a random variable can also be generated by subtracting from n the value of a binomial $(n, 1-p)$ random variable (why is that?). Hence, when $p > \frac{1}{2}$, we can generate a binomial $(n, 1-p)$ random variable by the above method and subtract its value from n to obtain the desired generation.

Remarks

1. Another way of generating a binomial (n, p) random variable X is by utilizing its interpretation as the number of successes in n independent Bernoulli trials, when each trial is a success with probability p. Consequently, we can also simulate X by generating the outcomes of these n Bernoulli trials.
2. As in the Poisson case, when the mean np is large it is better to first determine if the generated value is less than or equal to $I \equiv \text{Int}(np)$ or whether it is greater than I. In the former case one should then start the search with I, then $I-1, \ldots$, and so on; whereas in the latter case one should start searching with $I + 1$ and go upward. □

4.4 The Acceptance–Rejection Technique

Suppose we have an efficient method for simulating a random variable having probability mass function $\{q_j,\ j \geq 0\}$. We can use this as the basis for simulating from the distribution having mass function $\{p_j,\ j \geq 0\}$ by first simulating a random variable Y having mass function $\{q_j\}$ and then accepting this simulated value with a probability proportional to p_Y/q_Y.

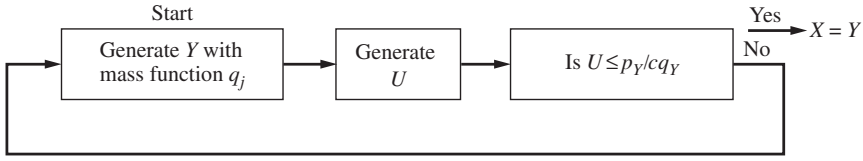

Figure 4.1. Acceptance–rejection.

Specifically, let c be a constant such that

$$\frac{p_j}{q_j} \le c \quad \text{for all } j \text{ such that } p_j > 0 \tag{4.2}$$

We now have the following technique, called the rejection method or the acceptance–rejection method, for simulating a random variable X having mass function $p_j = P\{X = j\}$.

Rejection Method

STEP 1: Simulate the value of Y, having probability mass function q_j.
STEP 2: Generate a random number U.
STEP 3: If $U < p_Y/cq_Y$, set $X = Y$ and stop. Otherwise, return to Step 1.

The rejection method is pictorially represented in Figure 4.1.

We now prove that the rejection method works.

Theorem *The acceptance–rejection algorithm generates a random variable X such that*

$$P\{X = j\} = p_j, \quad j = 0, \ldots$$

In addition, the number of iterations of the algorithm needed to obtain X is a geometric random variable with mean c.

Proof To begin, let us determine the probability that a single iteration produces the accepted value j. First note that

$$\begin{aligned} P\{Y = j, \text{it is accepted}\} &= P\{Y = j\}P\{\text{accept}|Y = j\} \\ &= q_j \frac{p_j}{cq_j} \\ &= \frac{p_j}{c} \end{aligned}$$

Summing over j yields the probability that a generated random variable is accepted:

$$P\{\text{accepted}\} = \sum_j \frac{p_j}{c} = \frac{1}{c}$$

As each iteration independently results in an accepted value with probability $1/c$, we see that the number of iterations needed is geometric with mean c. Also,

$$\begin{aligned} P\{X = j\} &= \sum_n P\{j \text{ accepted on iteration } n\} \\ &= \sum_n (1 - 1/c)^{n-1} \frac{p_j}{c} \\ &= p_j \quad \square \end{aligned}$$

Remark The reader should note that the way in which we "accept the value Y with probability p_Y/cq_Y" is by generating a random number U and then accepting Y if $U \leq p_Y/cq_Y$.

Example 4f Suppose we wanted to simulate the value of a random variable X that takes one of the values $1, 2, \ldots, 10$ with respective probabilities 0.11, 0.12, 0.09, 0.08, 0.12, 0.10, 0.09, 0.09, 0.10, 0.10. Whereas one possibility is to use the inverse transform algorithm, another approach is to use the rejection method with q being the discrete uniform density on $1, \ldots, 10$. That is, $q_j = 1/10$, $j = 1, \ldots, 10$. For this choice of $\{q_j\}$ we can choose c by

$$c = \text{Max}\frac{p_j}{q_j} = 1.2$$

and so the algorithm would be as follows:

STEP 1: Generate a random number U_1 and set $Y = \text{Int}(10U_1) + 1$.
STEP 2: Generate a second random number U_2.
STEP 3: If $U_2 \leq p_Y/.12$, set $X = Y$ and stop. Otherwise return to Step 1.

The constant 0.12 in Step 3 arises since $cq_Y = 1.2/10 = 0.12$. On average, this algorithm requires only 1.2 iterations to obtain the generated value of X. $\square$

The power of the rejection method, a version of which was initially proposed by the famous mathematician John von Neumann, will become even more readily apparent when we consider its analogue when generating continuous random variables.

4.5 The Composition Approach

Suppose that we had an efficient method to simulate the value of a random variable having either of the two probability mass functions $\{p_j^{(1)}, j \geq 0\}$ or $\{p_j^{(2)}, j \geq 0\}$, and that we wanted to simulate the value of the random variable X having mass function

$$P\{X = j\} = \alpha p_j^{(1)} + (1 - \alpha) p_j^{(2)}, \quad j \geq 0 \tag{4.3}$$

where $0 < \alpha < 1$. One way to simulate such a random variable X is to note that if X_1 and X_2 are random variables having respective mass functions $\{p_j^{(1)}\}$ and $\{p_j^{(2)}\}$, then the random variable X defined by

$$X = \begin{cases} X_1 \text{ with probability } \alpha \\ X_2 \text{ with probability } 1 - \alpha \end{cases}$$

will have its mass function given by (4.3). From this it follows that we can generate the value of such a random variable by first generating a random number U and then generating a value of X_1 if $U < \alpha$ and of X_2 if $U > \alpha$.

Example 4g Suppose we want to generate the value of a random variable X such that

$$p_j = P\{X = j\} = \begin{cases} 0.05 \text{ for } j = 1, 2, 3, 4, 5 \\ 0.15 \text{ for } j = 6, 7, 8, 9, 10 \end{cases}$$

By noting that $p_j = 0.5p_j^{(1)} + 0.5p_j^{(2)}$, where

$$p_j^{(1)} = 0.1, \quad j = 1, \ldots, 10 \quad \text{and} \quad p_j^{(2)} = \begin{cases} 0 \quad \text{for } j = 1, 2, 3, 4, 5 \\ 0.2 \text{ for } j = 6, 7, 8, 9, 10 \end{cases}$$

we can accomplish this by first generating a random number U and then generating from the discrete uniform over $1, \ldots, 10$ if $U < 0.5$ and from the discrete uniform over 6, 7, 8, 9, 10 otherwise. That is, we can simulate X as follows:

STEP 1: Generate a random number U_1.
STEP 2: Generate a random number U_2.
STEP 3: If $U_1 < 0.5$, set $X = \text{Int}(10U_2) + 1$. Otherwise,
set $X = \text{Int}(5U_2) + 6$. □

If $F_i, i = 1, \ldots, n$ are distribution functions and $\alpha_i, i = 1, \ldots, n$, are non negative numbers summing to 1, then the distribution function F given by

$$F(x) = \sum_{i=1}^{n} \alpha_i F_i(x)$$

is said to be a *mixture*, or a *composition*, of the distribution functions F_i, $i = 1, \ldots, n$. One way to simulate from F is first to simulate a random variable I, equal to i with probability $\alpha_i, i = 1, \ldots, n$, and then to simulate from the distribution F_I. (That is, if the simulated value of I is $I = j$, then the second simulation is from F_j.) This approach to simulating from F is often referred to as the *composition method.*

4.6 The Alias Method for Generating Discrete Random Variables

In this section we study a technique for generating discrete random variables which, although requiring some setup time, is very fast to implement.

In what follows, the quantities $\mathbf{P}, \mathbf{P}^{(k)}, \mathbf{Q}^{(k)}, k \leq n-1$, represent probability mass functions on the integers $1, 2, \ldots, n$—that is, they are n-vectors of nonnegative numbers summing to 1. In addition, the vector $\mathbf{P}^{(k)}$ has at most k nonzero components, and each of the $\mathbf{Q}^{(k)}$ has at most two nonzero components. We show that any probability mass function $\mathbf{P}$ can be represented as an equally weighted mixture of $n-1$ probability mass functions $\mathbf{Q}$ (each having at most two nonzero components). That is, we show, for suitably defined $\mathbf{Q}^{(1)}, \ldots, \mathbf{Q}^{(n-1)}$, that $\mathbf{P}$ can be expressed as

$$\mathbf{P} = \frac{1}{n-1} \sum_{k=1}^{n-1} \mathbf{Q}^{(k)} \tag{4.4}$$

As a prelude to presenting the method for obtaining this representation, we need the following simple lemma whose proof is left as an exercise.

Lemma Let $P = \{P_i, i = 1, \ldots, n\}$ denote a probability mass function. Then

(a) there exists an i, $1 \leq i \leq n$, *such that* $P_i < 1/(n-1)$, and
(b) for this i there exists aj, $j \neq i$, such that $P_i + P_j \geq 1/(n-1)$.

Before presenting the general technique for obtaining the representation (4.4), let us illustrate it by an example.

Example 4h Consider the three-point distribution $\mathbf{P}$ with $P_1 = \frac{7}{16}$, $P_2 = \frac{1}{2}$, $P_3 = \frac{1}{16}$. We start by choosing i and j satisfying the conditions of the preceding lemma. Since $P_3 < \frac{1}{2}$ and $P_3 + P_2 \geq \frac{1}{2}$, we can work with $i = 3$ and $j = 2$. We now define a two-point mass function $\mathbf{Q}^{(1)}$, putting all its weight on 3 and 2 and such that $\mathbf{P}$ is expressible as an equally weighted mixture between $\mathbf{Q}^{(1)}$ and a second two-point mass function $\mathbf{Q}^{(2)}$. In addition, all the mass of point 3 is contained in $\mathbf{Q}^{(1)}$. As we have

$$P_j = \frac{1}{2}(Q_j^{(1)} + Q_j^{(2)}), \quad j = 1, 2, 3 \tag{4.5}$$

and $Q_3^{(2)}$ is supposed to equal 0, we must therefore take

$$Q_3^{(1)} = 2P_3 = \frac{1}{8}, \qquad Q_2^{(1)} = 1 - Q_3^{(1)} = \frac{7}{8}, \qquad Q_1^{(1)} = 0$$

To satisfy (10.2), we must then set

$$Q_3^{(2)} = 0, \qquad Q_2^{(2)} = 2P_2 - \frac{7}{8} = \frac{1}{8}, \qquad Q_1^{(2)} = 2P_1 = \frac{7}{8}$$

Hence we have the desired representation in this case. Suppose now that the original distribution was the following four-point mass function:

$$P_1 = \frac{7}{16}, \qquad P_2 = \frac{1}{4}, \qquad P_3 = \frac{1}{8}, \qquad P_4 = \frac{3}{16}$$

Now $P_3 < \frac{1}{3}$ and $P_3 + P_1 \geq \frac{1}{3}$. Hence our initial two-point mass function —$\mathbf{Q}^{(1)}$ — concentrates on points 3 and 1 (giving no weight to 2 and 4). Because the final representation gives weight $\frac{1}{3}$ to $\mathbf{Q}^{(1)}$ and in addition the other $\mathbf{Q}^{(j)}$, $j = 2, 3$, do not give any mass to the value 3, we must have that

$$\frac{1}{3} Q_3^{(1)} = P_3 = \frac{1}{8}$$

Hence

$$Q_3^{(1)} = \frac{3}{8}, \qquad Q_1^{(1)} = 1 - \frac{3}{8} = \frac{5}{8}$$

Also, we can write

$$\mathbf{P} = \frac{1}{3}\mathbf{Q}^{(1)} + \frac{2}{3}\mathbf{P}^{(3)}$$

where $\mathbf{P}^{(3)}$, to satisfy the above, must be the vector

$$\begin{aligned}
\mathbf{P}_1^{(3)} &= \frac{3}{2}\left(P_1 - \frac{1}{3}Q_1^{(1)}\right) = \frac{11}{32} \\
\mathbf{P}_2^{(3)} &= \frac{3}{2}P_2 = \frac{3}{8} \\
\mathbf{P}_3^{(3)} &= 0 \\
\mathbf{P}_4^{(3)} &= \frac{3}{2}P_4 = \frac{9}{32}
\end{aligned}$$

Note that $\mathbf{P}^{(3)}$ gives no mass to the value 3. We can now express the mass function $\mathbf{P}^{(3)}$ as an equally weighted mixture of two-point mass functions $\mathbf{Q}^{(2)}$ and $\mathbf{Q}^{(3)}$, and we end up with

$$\begin{aligned}
\mathbf{P} &= \frac{1}{3}\mathbf{Q}^{(1)} + \frac{2}{3}\left(\frac{1}{2}\mathbf{Q}^{(2)} + \frac{1}{2}\mathbf{Q}^{(3)}\right) \\
&= \frac{1}{3}(\mathbf{Q}^{(1)} + \mathbf{Q}^{(2)} + \mathbf{Q}^{(3)})
\end{aligned}$$

(We leave it as an exercise for the reader to fill in the details.) □

The above example outlines the following general procedure for writing the n-point mass function $\mathbf{P}$ in the form (4.4), where each of the $\mathbf{Q}^{(i)}$ are mass functions giving all their mass to at most two points. To start, we choose i and j satisfying the conditions of the lemma. We now define the mass function $\mathbf{Q}^{(1)}$ concentrating

on the points i and j and which contain all the mass for point i by noting that in the representation (4.4) $Q_i^{(k)} = 0$ for $k = 2, \ldots, n-1$, implying that

$$Q_i^{(1)} = (n-1)P_i \qquad \text{and so } Q_j^{(1)} = 1 - (n-1)P_i$$

Writing

$$\mathbf{P} = \frac{1}{n-1}\mathbf{Q}^{(1)} + \frac{n-2}{n-1}\mathbf{P}^{(n-1)} \tag{4.6}$$

where $\mathbf{P}^{(n-1)}$ represents the remaining mass, we see that

$$\begin{aligned}
P_i^{(n-1)} &= 0 \\
P_j^{(n-1)} &= \frac{n-1}{n-2}\left(P_j - \frac{1}{n-1}Q_j^{(1)}\right) = \frac{n-1}{n-2}\left(P_i + P_j - \frac{1}{n-1}\right) \\
P_k^{(n-1)} &= \frac{n-1}{n-2}P_k, \quad k \neq i \text{ or } j
\end{aligned}$$

That the above is indeed a probability mass function is easily checked—for example, the nonnegativity of $P_j^{(n-1)}$ follows from the fact that j was chosen so that $P_i + P_j \geq 1/(n-1)$.

We may now repeat the above procedure on the $(n-1)$ point probability mass function $\mathbf{P}^{(n-1)}$ to obtain

$$\mathbf{P}^{(n-1)} = \frac{1}{n-2}\,\mathbf{Q}^{(2)} + \frac{n-3}{n-2}\mathbf{P}^{(n-2)}$$

and thus from (4.6) we have

$$\mathbf{P} = \frac{1}{n-1}\,\mathbf{Q}^{(1)} + \frac{1}{n-1}\,\mathbf{Q}^{(2)} + \frac{n-3}{n-1}\,\mathbf{P}^{(n-2)}$$

We now repeat the procedure on $P^{(n-2)}$ and so on until we finally obtain

$$\mathbf{P} = \frac{1}{n-1}(\mathbf{Q}^{(1)} + \cdots + \mathbf{Q}^{(n-1)})$$

In this way we are able to represent $\mathbf{P}$ as an equally weighted mixture of $n-1$ two-point mass functions. We can now easily simulate from $\mathbf{P}$ by first generating a random integer N equally likely to be either $1, 2, \ldots, n-1$. If the resulting value N is such that $\mathbf{Q}^{(N)}$ puts positive weight only on the points i_N and j_N, we can set X equal to i_N if a second random number is less than $\mathbf{Q}_{i_N}^{(N)}$ and equal to j_N otherwise. The random variable X will have probability mass function $\mathbf{P}$. That is, we have the following procedure for simulating from $\mathbf{P}$.

STEP 1: Generate U_1 and set $N = 1 + \text{Int}\,[(n-1)U_1]$.
STEP 2: Generate U_2 and set

$$X = \begin{cases} i_N & \text{if } U_2 < Q_{i_N}^{(N)} \\ j_N & \text{otherwise} \end{cases}$$

Remarks

1. The above is called the alias method because by a renumbering of the $\mathbf{Q}$'s we can always arrange things so that for each k, $Q_k^{(k)} > 0$. (That is, we can arrange things so that the kth two-point mass function gives positive weight to the value k.) Hence, the procedure calls for simulating N, equally likely to be $1, 2, \ldots, n-1$, and then if $N = k$ it either accepts k as the value of X, or it accepts for the value of X the "alias" of k (namely, the other value that $\mathbf{Q}^{(k)}$ gives positive weight).
2. Actually, it is not necessary to generate a new random number in Step 2. Because $N-1$ is the integer part of $(n-1)U_1$, it follows that the remainder $(n-1)U_1 - (N-1)$ is independent of N_1 and is uniformly distributed on $(0, 1)$. Hence, rather than generating a new random number U_2 in Step 2, we can use $(n-1)U_1 - (N-1)$. □

4.7 Generating Random Vectors

A random vector $X_1, \ldots, X_n$ can be simulated by sequentially generating the X_i. That is, first generate X_1; then generate X_2 from its conditional distribution given the generated value of X_1; then generate X_3 from its conditional distribution given the generated values of X_1 and X_2; and so on. This is illustrated in Example 4i, which shows how to simulate a random vector having a multinomial distribution.

Example 4i Consider n independent trials, each of which results in one of the outcomes $1, 2, \ldots, r$ with respective probabilities $p_1, p_2, \ldots, p_r$, $\sum_{i=1}^{r} p_i = 1$. If X_i denotes the number of trials that result in outcome i, then the random vector $(X_1, \ldots, X_r)$ is said to be a multinomial random vector. Its joint probability mass function is given by

$$P\{X_i = x_i, i = 1, \ldots, r\} = \frac{n!}{x_1! \cdots x_r!} p_1^{x_1} \cdots p_r^{x_r}, \quad \sum_{i=1}^{r} x_i = n$$

The best way to simulate such a random vector depends on the relative sizes of r and n. If r is large relative to n, so that many of the outcomes do not occur on any of the trials, then it is probably best to simulate the random variables by generating

the outcomes of the n trials. That is, first generate independent random variables $Y_1, \ldots, Y_n$ such that

$$P\{Y_j = i\} = p_i, \quad i = 1, \ldots, r, \quad j = 1, \ldots, n,$$

and then set

$$X_i = \text{number of } j, \quad j = 1, \ldots, n\text{: } Y_j = i$$

(That is, the generated value of Y_j represents the result of trial j, and X_i is the number of trials that result in outcome i.)

On the other hand, if n is large relative to r, then $X_1, \ldots, X_r$ can be simulated in sequence. That is, first generate X_1, then X_2, then X_3, and so on. Because each of the n trials independently results in outcome 1 with probability p_1, it follows that X_1 is a binomial random variable with parameters (n, p_1). Therefore, we can use the method of Section 4.3 to generate X_1. Suppose its generated value is x_1. Then, given that x_1 of the n trials resulted in outcome 1, it follows that each of the other $n - x_1$ trials independently results in outcome 2 with probability

$$P\{2|\text{not1}\} = \frac{p_2}{1 - p_1}$$

Therefore, the conditional distribution of X_2, given that $X_1 = x_1$, is binomial with parameters $(n - x_1, \frac{p_2}{1-p_1})$. Thus, we can again make use of Section 4.3 to generate the value of X_2. If the generated value of X_2 is x_2, then we next need to generate the value of X_3 conditional on the results that $X_1 = x_1$, $X_2 = x_2$. However, given there are x_1 trials that result in outcome 1 and x_2 trials that result in outcome 2, each of the remaining $n - x_1 - x_2$ trials independently results in outcome 3 with probability $\frac{p_3}{1-p_1-p_2}$. Consequently, the conditional distribution of X_3 given that $X_i = x_i$, $i = 1, 2$, is binomial with parameters $(n - x_1 - x_2, \frac{p_3}{1-p_1-p_2})$. We then use this fact to generate X_3, and continue on until all the values $X_1, \ldots, X_r$ have been generated. □

Exercises

1. Write a program to generate n values from the probability mass function $p_1 = \frac{1}{3}$, $p_2 = \frac{2}{3}$.

(a) Let $n = 100$, run the program, and determine the proportion of values that are equal to 1.
(b) Repeat (a) with $n = 1000$.
(c) Repeat (a) with $n = 10{,}000$.

2. Write a computer program that, when given a probability mass function $\{p_j, j = 1, \ldots, n\}$ as an input, gives as an output the value of a random variable having this mass function.

3. Give an efficient algorithm to simulate the value of a random variable X such that

$$P\{X = 1\} = 0.3,\ P\{X = 2\} = 0.2,\ P\{X = 3\} = 0.35,\ P\{X = 4\} = 0.15$$

4. A deck of 100 cards—numbered $1, 2, \ldots, 100$—is shuffled and then turned over one card at a time. Say that a "hit" occurs whenever card i is the ith card to be turned over, $i = 1, \ldots, 100$. Write a simulation program to estimate the expectation and variance of the total number of hits. Run the program. Find the exact answers and compare them with your estimates.

5. Another method of generating a random permutation, different from the one presented in Example 4b, is to successively generate a random permutation of the elements $1, 2, \ldots, n$ starting with $n = 1$, then $n = 2$, and so on. (Of course, the random permutation when $n = 1$ is 1.) Once one has a random permutation of the first $n - 1$ elements—call it $P_1, \ldots, P_{n-1}$—the random permutation of the n elements $1, \ldots, n$ is obtained by putting n in the final position—to obtain the permutation $P_1, \ldots, P_{n-1}, n$—and then interchanging the element in position n (namely, n) with the element in a randomly chosen position which is equally likely to be either position 1, position 2, ..., or position n.

(a) Write an algorithm that accomplishes the above.
(b) Prove by mathematical induction on n that the algorithm works, in that the permutation obtained is equally likely to be any of the $n!$ permutations of $1, 2, \ldots, n$.

6. Using an efficient procedure, along with the text's random number sequence, generate a sequence of 25 independent Bernoulli random variables, each having parameter $p = .8$. How many random numbers were needed?

7. A pair of fair dice are to be continually rolled until all the possible outcomes $2, 3, \ldots, 12$ have occurred at least once. Develop a simulation study to estimate the expected number of dice rolls that are needed.

8. Suppose that each item on a list of n items has a value attached to it, and let $v(i)$ denote the value attached to the ith item on the list. Suppose that n is very large, and also that each item may appear at many different places on the list. Explain how random numbers can be used to estimate the sum of the values of the different items on the list (where the value of each item is to be counted once no matter how many times the item appears on the list).

9. Consider the n events $A_1, \ldots, A_n$ where A_i consists of the following n_i outcomes: $A_i = \{a_{i,1}, a_{i,2}, \ldots, a_{i,n_i}\}$. Suppose that for any given outcome a, $P\{a\}$, the probability that the experiment results in outcome a is known.

Explain how one can use the results of Exercise 8 to estimate $P\{\bigcup_{i=1}^{n} A_i\}$, the probability that at least one of the events A_i occurs. Note that the events A_i, $i = 1, \ldots, n$, are not assumed to be mutually exclusive.

10. The negative binomial probability mass function with parameters (r, p), where r is a positive integer and $0 < p < 1$, is given by

$$p_j = \frac{(j-1)!}{(j-r)!(r-1)!} p^r (1-p)^{j-r}, \quad j = r, r+1, \ldots$$

(a) Use the relationship between negative binomial and geometric random variables and the results of Example 4d to obtain an algorithm for simulating from this distribution.
(b) Verify the relation

$$p_{j+1} = \frac{j(1-p)}{j+1-r} p_j$$

(c) Use the relation in part (b) to give a second algorithm for generating negative binomial random variables.
(d) Use the interpretation of the negative binomial distribution as the number of trials it takes to amass a total of r successes when each trial independently results in a success with probability p, to obtain still another approach for generating such a random variable.

11. Give an efficient method for generating a random subset of size r from the set $\{1, \ldots, n\}$ conditional on the event that the subset contains at least one of the elements of $\{1, \ldots, k\}$ when r and k much smaller than n.

12. If Z is a standard normal random variable, show that

$$E[|Z|] = \left(\frac{2}{\pi}\right)^{1/2} \approx 0.798$$

13. Give two methods for generating a random variable X such that

$$P\{X = i\} = \frac{e^{-\lambda}\lambda^i / i!}{\sum_{j=0}^{k} e^{-\lambda}\lambda^j / j!}, \quad i = 0, \ldots, k$$

14. Let X be a binomial random variable with parameters n and p. Suppose that we want to generate a random variable Y whose probability mass function is the same as the conditional mass function of X given that $X \geq k$, for some $k \leq n$. Let $\alpha = P\{X \geq k\}$ and suppose that the value of α has been computed.

(a) Give the inverse transform method for generating Y.
(b) Give a second method for generating Y.

(c) For what values of α, small or large, would the algorithm in (b) be inefficient?

15. Give a method for simulating X, having the probability mass function p_j, $j = 5, 6, \ldots, 14$, where

$$p_j = \begin{cases} 0.11 \text{ when } j \text{ is odd and } 5 \le j \le 13 \\ 0.09 \text{ when } j \text{ is even and } 6 \le j \le 14 \end{cases}$$

Use the text's random number sequence to generate X.

16. Suppose that the random variable X can take on any of the values $1, \ldots, 10$ with respective probabilities 0.06, 0.06, 0.06, 0.06, 0.06, 0.15, 0.13, 0.14, 0.15, 0.13. Use the composition approach to give an algorithm that generates the value of X. Use the text's random number sequence to generate X.

17. Present a method to generate the value of X, where

$$P\{X = j\} = \left(\frac{1}{2}\right)^{j+1} + \frac{\left(\frac{1}{2}\right) 2^{j-1}}{3^j}, \quad j = 1, 2, \ldots$$

Use the text's random number sequence to generate X.

18. Let X have mass function $p_j = P\{X = j\}$, $\sum_{j=1}^{\infty} p_j = 1$. Let

$$\lambda_n = P\{X = n | X > n - 1\} = \frac{p_n}{1 - \sum_{j=1}^{n-1} p_j}, \quad n = 1, \ldots$$

(a) Show that $p_1 = \lambda_1$ and

$$p_n = (1 - \lambda_1)(1 - \lambda_2) \cdots (1 - \lambda_{n-1})\lambda_n$$

The quantities λ_n, $n \ge 1$, are called the discrete hazard rates, since if we think of X as the lifetime of some item then λ_n represents the probability that an item that has reached the age n will die during that time period. The following approach to simulating discrete random variables, called the discrete hazard rate method, generates a succession of random numbers, stopping when the nth random number is less than λ_n. The algorithm can be written as follows:

STEP 1: $X = 1$.
STEP 2: Generate a random number U.
STEP 3: If $U < \lambda_X$, stop.
STEP 4: $X = X + 1$.
STEP 5: Go to Step 2.

(a) Show that the value of X when the above stops has the desired mass function.
(b) Suppose that X is a geometric random variable with parameter p. Determine the values $\lambda_n, n \geq 1$. Explain what the above algorithm is doing in this case and why its validity is clear.

19. Suppose that $0 \leq \lambda_n \leq \lambda$, for all $n \geq 1$. Consider the following algorithm to generate a random variable having discrete hazard rates $\{\lambda_n\}$.

STEP 1: $S = 0$.
STEP 2: Generate U and set $Y = \text{Int}\left(\frac{\log(U)}{\log(1-\lambda)}\right) + 1$.
STEP 3: $S = S + Y$.
STEP 4: Generate U.
STEP 5: If $U \leq \lambda_S/\lambda$, set $X = S$ and stop. Otherwise, go to 2.

(a) What is the distribution of Y in Step 2?
(b) Explain what the algorithm is doing.
(c) Argue that X is a random variable with discrete hazard rates $\{\lambda_n\}$.

20. Suppose X and Y are discrete random variables and that you want to generate the value of a random variable W with probability mass function

$$P(W = i) = P(X = i|Y = j)$$

for some specified j for which $P(Y = j) > 0$. Show that the following algorithm accomplishes this.

(a) Generate the value of a random variable having the distribution of X.
(b) Let i being the generated value in (a).
(c) Generate a random number U.
(d) If $U < P(Y = j|X = i)$ set $W = i$ and stop.
(e) Return to (a).

21. Set up the alias method for generating a binomial with parameters (5, 0.4).

22. Explain how we can number the $\mathbf{Q}^{(k)}$ in the alias methodso that k is one of the two points to which $\mathbf{Q}^{(k)}$ gives weight.

23. A random selection of m balls is to be made from an urn that contains n balls, n_i of which have color type i, $\sum_{i=1}^{r} n_i = n$. Discuss efficient procedures for simulating $X_1, \ldots, X_r$, where X_i denotes the number of withdrawn balls that have color type i.