Simulation, 3rd Edition

Solution of Exercise Problems

Yan Zeng

September 3, 2008

Contents

This is a solution manual for the textbook *Simulation*, 3rd Edition, by Sheldon M. Ross (2002, Academic Press). This version omits the Problem 8, 9, 18 of Chapter 4.

Chapter 1

Introduction

1. (a)

Proof. The Matlab code for the problem solution is given below.

```
function departureTimes = ross_1_1a(arrivalTimes, serviceTimes)

%ROSS_1_1a  solves Exerciese Problem 1(a) of Chapter 1, [1].
%
%           Reference:
%           [1] Ross: Simulation, Third Edition, Academic Press, 2002.

if (numel(arrivalTimes)~= numel(serviceTimes))
    error('Array of arrival times and array of service times must have same size');
end

numCustomers = numel(arrivalTimes);
serverEmptyTime = 0;               % The time when the server is empty.
departureTimes = zeros(1, numCustomers);

for i=1:numCustomers
    if (serverEmptyTime < arrivalTimes(i))
        departureTimes(i) = arrivalTimes(i) + serviceTimes(i);
    else
        departureTimes(i) = serverEmptyTime + serviceTimes(i);
    end
    serverEmptyTime = departureTimes(i);
end
```

□

(b)

Proof. The Matlab code for the problem solution is given below.

```
function departureTimes = ross_1_1b(arrivalTimes, serviceTimes)

%ROSS_1_1b  solves Exerciese Problem 1(b) of Chapter 1, [1].
%
%           Reference:
%           [1] Ross: Simulation, Third Edition, Academic Press, 2002.
```

```
if (numel(arrivalTimes)~= numel(serviceTimes))
    error('Array of arrival times and array of service times must have same size');
end

numCustomers = numel(arrivalTimes);
numServers = 2;
serverEmptyTimes = zeros(1, numServers); % serverEmptyTimes(i) is the time
                                         % when the i-th server is empty.
departureTimes = zeros(1, numCustomers);

for i=1:numCustomers
    % serverTime is the earliest time when a server is empty; serverNo is
    % this server's number (1 or 2).
    [serverTime, serverNo] = min(serverEmptyTimes);

    if (serverTime < arrivalTimes(i))
        departureTimes(i) = arrivalTimes(i) + serviceTimes(i);
    else
        departureTimes(i) = serverTime + serviceTimes(i);
    end
    serverEmptyTimes(serverNo) = departureTimes(i);
end
```

□

(c)

Proof. The Matlab code for the problem solution is given below.

```
function departureTimes = ross_1_1c(arrivalTimes, serviceTimes)

%ROSS_1_1c  solves Exerciese Problem 1(c) of Chapter 1, [1].
%
%           Reference:
%           [1] Ross: Simulation, Third Edition, Academic Press, 2002.

if (numel(arrivalTimes)~= numel(serviceTimes))
    error('Array of arrival times and array of service times must have same size');
end

numCustomers = numel(arrivalTimes);
serverEmptyTime = 0; % The time when the server is empty
departureTimes = zeros(1, numCustomers);
unserved = ones(1, numCustomers); % unserved(i) has value 1 if the i-th customer has
                                  % not been served, and value 0 otherwise.

while ( sum(unserved) > 0 ) % sum(unserved)>0 iff there are customers unserved.

    % Find the customer who has been waiting the least time. If no customer
    % is waiting, get the next customer to arrive.
    waitingCustomers = ( (arrivalTimes <= serverEmptyTime) & unserved );

    if ( sum(waitingCustomers) == 0 ) % There are no customers waiting.
        [temp, next] = max(arrivalTimes > serverEmptyTime);
        departureTimes(next) = arrivalTimes(next) + serviceTimes(next);
```

```
    else % There are customers waiting.
        [temp, next] = max(cumsum(waitingCustomers));
        departureTimes(next) = serverEmptyTime + serviceTimes(next);
    end
    serverEmptyTime = departureTimes(next);
    unserved(next) = 0;
end
```

□

2.

Proof. When there are k servers, the relation is given by $D_n - S_n = \max\{A_n, \min\{D_{n-1}, D_{n-2}, \cdots, D_{n-k}\}\}$, where $D_0 = D_{-1} = \cdots = D_{-(k-1)} = 0$. The corresponding Matlab code is given below.

```
function departureTimes = ross_1_2(arrivalTimes, serviceTimes, numServers)

%ROSS_1_2  solves Exerciese Problem 2 of Chapter 1, [1].
%
%           Reference:
%           [1] Ross: Simulation, Third Edition, Academic Press, 2002.

if (numel(arrivalTimes)~= numel(serviceTimes))
    error('Array of arrival times and array of service times must have same size');
end

numCustomers = numel(arrivalTimes);
serverEmptyTimes = zeros(1, numServers); % serverEmptyTimes(i) is the time when
                                         % the i-th server is empty.
departureTimes = zeros(1, numCustomers);

for i=1:numCustomers
    % serverTime is the earliest time when a server is empty; serverNo is
    % this server's number (1, 2, ..., k).
    [serverTime, serverNo] = min(serverEmptyTimes);

    if (serverTime < arrivalTimes(i))
        departureTimes(i) = arrivalTimes(i) + serviceTimes(i);
    else
        departureTimes(i) = serverTime + serviceTimes(i);
    end
    serverEmptyTimes(serverNo) = departureTimes(i);
end
```

□

Chapter 2

Elements of Probability

1. (a)

Proof. $B = (A \cup A^c) \cap B = AB \cup A^cB$. $A \cup B = A \cup (AB \cup A^cB) = (A \cup AB) \cup A^cB = A \cup A^cB$. □

(b)

Proof. $P(A \cup B) = P(A \cup A^cB) = P(A) + P(A^cB) = P(A) + P(B - AB) = P(A) + P(B) - P(AB)$. □

2.

Proof. $A \cup B$ can be divided into the disjoint union of three sets. The first set consists of all the 6-tuples whose second entry is 2. This set has $5! = 120$ elements. This second set consists of all the 6-tuples whose second entry is 1. So it has $5! = 120$ elements. The third set consists all the 6-tuples where 1 occupies either the first entry or the second entry and 2 does not occupy the second entry. It has $2 \cdot 4 \cdot 4! = 192$ elements. So $\#AB = 432$.
$AB = \{(1, 2, i, j, k, l), (i, 2, 1, j, k, l) : \{i, j, k, l\} = \{3, 4, 5, 6\}\}$. So $\#AB = 2 \cdot 4! = 48$.
$ABC = \{(1, 2, 3, i, j, k) : \{i, j, k\} = \{4, 5, 6\}\}$. So $\#ABC = 3! = 6$.
Finally, $\#A \cup BC = \#A + \#BC - ABC = 3 \cdot 5! + 4! - 3! = 360 + 24 - 6 = 378$. □

3.

Proof. By assumption, $P(bb) = P(bg) = P(gb) = P(gg) = \frac{1}{4}$. So

$$P(gg|gg \text{ or } gb) = \frac{P(gg)}{P(gg) + P(gb)} = \frac{1}{2}.$$

Remark: The problem is essentially the same as the coin-flipping problem at the beginning of §2.3. □

4.

Proof. We assume $P(bb) = P(bg) = P(gb) = P(gg) = \frac{1}{4}$. Then the desired probability equals to

$$P(bb|bb \text{ or } bg \text{ } gb) = \frac{1}{3}.$$

Remark: For some subtle issues related to conditional probability, see Feller [2], V.1, Example (c). □

5.

Proof. Since $1 = \sum_{i=1}^{4} P(X = i) = (1 + 2 + 3 + 4)c$, we conclude $c = 10$. So $P(2 \le X \le 3) = (2 + 3)/10 = 0.5$. □

6.

Proof. Since $1 = \int_0^1 f(x)dx = \frac{cx^2}{2}|_0^1 = \frac{c}{2}$, we conclude $c = 2$. So $P(X > \frac{1}{2}) = \int_{\frac{1}{2}}^1 2xdx = x^2|_{\frac{1}{2}}^1 = \frac{3}{4}$. □

7.

Proof.

$$\begin{aligned} P(X < Y) &= \int_0^\infty \int_0^\infty 1_{\{x<y\}} 2e^{-(x+2y)} dxdy = \int_0^\infty 2e^{-2y} dy \int_0^y e^{-x} dx \\ &= \int_0^\infty 2e^{-2y}(1 - e^{-y})dy = \int_0^\infty 2e^{-2y} dy - \int_0^\infty 2e^{-3y} dy \\ &= 1 - \frac{2}{3} = \frac{1}{3}. \end{aligned}$$

□

8.

Proof. $E[X] = \sum_{i=1}^4 \frac{i^2}{10} = 3$. □

9.

Proof. $E[X] = \int_0^1 2x^2 dx = \frac{2}{3}$. □

10.

Proof. Following the hint, we have $E[X] = \sum_{i=1}^{10} E[X_i]$. Note for each i,

$$P(X_i = 1) = 1 - P(\text{type i coupon is not among the } N \text{ coupons}) = 1 - \frac{9^N}{10^N} = 1 - 0.9^N.$$

So $E[X] = 10 \cdot (1 - 0.9^N)$. □

11.

Proof. $P(X = i) = \frac{1}{6}$, $i = 1, \cdots, 6$. So $E[X] = \frac{1}{6}\sum_{i=1}^6 i = 3.5$ and $E[X^2] = \frac{1}{6}\sum_{i=1}^6 i^2 = \frac{1}{6}\frac{6\cdot(6+1)\cdot(2\cdot6+1)}{6} = \frac{91}{6}$. Therefore $Var(X) = \frac{91}{6} - 3.5^2 \approx 2.92$. □

12.

Proof. Since $1 = \int_0^1 f(x)dx = c(e-1)$, we conclude $c = \frac{1}{e-1}$. We have

$$E[X] = c\int_0^1 xe^x dx = c\left(xe^x|_0^1 - \int_0^1 e^x dx\right) = c$$

and

$$E[X^2] = c\int_0^1 x^2 e^x dx = c\left(x^2 e^x|_0^1 - 2\int_0^1 xe^x dx\right) = ce - 2E[X] = ce - c.$$

So $Var(X) = E[X^2] - (EX)^2 = ce - c - c^2 = \frac{1}{e-1}(e - 1 - \frac{1}{e-1}) = \frac{(e-1)^2-1}{(e-1)^2}$. □

13.

Proof. $Var(aX+b) = E(aX+b-E(aX+b))^2 = E(aX+b-aEX-b)^2 = a^2E(X-EX)^2 = a^2Var(X)$. □

14.

Proof. The answer of the problem depends on what distribution the amount of liquid apple follows. For example, if we assume it follows a normal distribution, then the problem's conditions imply mean $\mu = 4$ and variance $\sigma^2 = 4$. Denote by X the amount of liquid apple, then

$$P(X \geq 6) = P\left(\frac{X-4}{2} \geq 1\right) = 0.1587, P(3 \leq X \leq 5) = P\left(-0.5 \leq \frac{X-4}{2} \leq 0.5\right) = 0.3829.$$

□

15.

Proof. The probability that a three-engine plane functions is $p_1 = \binom{3}{2} p^2(1-p) + p^3 = 3p^2 - 2p^3$. The probability that a five-engine plane functions is $p_2 = \binom{5}{3} p^3(1-p)^2 + \binom{5}{4} p^4(1-p) + p^5 = p^3(6p^2 - 15p + 10)$. So $p_1 \geq p_2$ is reduced to $0 \geq (p-1)^2(2p-1)$, i.e. $p \leq 0.5$. □

16.

Proof. For $i = 1, \cdots, n$,

$$\frac{P(X=i)}{P(X=i-1)} = \frac{\binom{n}{i} p^i (1-p)^{n-i}}{\binom{n}{i-1} p^{i-1}(1-p)^{n-i+1}} = \frac{n-i+1}{i} \cdot \frac{p}{1-p}.$$

So $P(X = i)$ is an increasing function of i if and only if $(n-i+1)p \geq i(1-p)$, which is equivalent to $(n+1)p \geq i$. This shows $P(X = i)$ first increases and then decreases, reaching its maximum value when i is the largest integer less than or equal to $(n+1)p$. □

17.

Proof. X is the sum of n i.i.d. Bernoulli random variables; Y is the sum of m i.i.d. Bernoulli random variables. Independence of X and Y implies these two families of Bernoulli random variables are also independent to each other. So $X + Y$ as the sum of $n + m$ i.i.d. Bernoulli random variables is binomial with parameters $(n + m, p)$. □

18.

Proof. Because Poisson random variables may be used to approximate the distribution of the number of successes in a large number of trials when each trial has a small probability of being a success. See page 18-19 for details. □

19.

Proof. We calculate the moment generating function of X:

$$F(u) = E[e^{Xu}] = \sum_{k=0}^{\infty} e^{uk} \frac{\lambda^k}{k!} e^{-\lambda} = e^{-\lambda} e^{\lambda e^u} = e^{\lambda(e^u - 1)}.$$

Then it's straightforward to calculate $F'(u) = \lambda e^{\lambda(e^u-1)+u}$ and $F''(u) = \lambda e^{\lambda(e^u-1)+u}(\lambda e^u + 1)$. So $E[X] = F'(0) = \lambda$, $E[X^2] = F''(0) = \lambda^2 + \lambda$, and $Var(X) = E[X^2] - (E[X])^2 = \lambda$. □

20.

Proof. X can be seen as the limit of a sequence of binomial random variables $(X_i)_{i=1}^{\infty}$, such that X_i has parameters (n_i, p_i) and $n_i p_i \to \lambda_1$ as $i \to \infty$.

Similarly, Y can be seen as the limit of a sequence of binomial random variables $(Y_i)_{i=1}^{\infty}$, such that Y_i has parameters (m_i, p_i) and $m_i p_i \to \lambda_2$ as $i \to \infty$.

By the result of Exercise 17, $X_i + Y_i$ is binomial with parameters $(m_i + n_i, p_i)$ such that $(m_i + n_i)p_i \to (\lambda_1 + \lambda_2)$. Therefore, $X + Y = \lim_{i\to\infty}(X_i + Y_i)$ is Poisson with parameter $\lambda_1 + \lambda_2$.

For an analytic proof, we note

$$\begin{aligned} P(X+Y=k) &= \sum_{i=0}^{k} P(X=i)P(Y=k-i) \\ &= \sum_{i=0}^{k} e^{-\lambda_1}\frac{\lambda_1^i}{i!} \cdot e^{-\lambda_2}\frac{\lambda_2^{k-i}}{(k-i)!} \\ &= \frac{e^{-(\lambda_1+\lambda_2)}}{k!}\sum_{i=0}^{k}\frac{k!}{i!(k-i)!}\lambda_1^i\lambda_2^{k-i} \\ &= \frac{e^{-(\lambda_1+\lambda_2)}}{k!}(\lambda_1+\lambda_2)^k. \end{aligned}$$

□

21.

Proof. We can utilize this recursion equations as follows:

```
function p = ross_2_21(lambda, n)

%ROSS_2_21  solves Exerciese Problem 21 of Chapter 2, [1]. The function
%           calculates the probability of a Poisson random variable with
%           rate LAMBDA being equal to N.
%
%           Reference:
%           [1] Ross: Simulation, Third Edition, Academic Press, 2002.

if (lambda <= 0) || (n < 0)
    error('Invalid argument');
end

p = exp(-lambda);
if (n==0)
    return;
else
    for i=0:(n-1)
        p = p * lambda / (i+1);
    end
end
```

□

22.

Proof. $P(X > n) = \sum_{k=n+1}^{\infty} p(1-p)^{k-1} = p(1-p)^n \sum_{k=0}^{\infty}(1-p)^k = p(1-p)^n \cdot \frac{1}{p} = (1-p)^n$. □

23.

Proof. Let $(X_i)_{i=1}^{\infty}$ be i.i.d. Bernoulli random variables with parameter $p = 0.6$, such that

$$X_i = \begin{cases} 1 & \text{if A wins the i-th match} \\ 0 & \text{otherwise.} \end{cases}$$

If we define $S_n = \sum_{i=1}^{n} X_i$, then A wins the match if and only if the random walk S_n reaches 5 before it reaches -5. This probability is $\frac{-(-5)}{5-(-5)} = 0.5$ by Wald's equation. For details, see Durrett [1], page 180, Example 1.5. [1] □

24.

Proof. $X = \sum_{i=1}^{n} Y_i$ is self-evident. By symmetry, Y_i has the same distribution, with $P(Y_i = 1) = \frac{N}{N+M}$. So $E[X] = \frac{nN}{N+M}$. □

25.

Proof. $P(5 \le U \le 15) = \frac{15-5}{30} = \frac{1}{3}$. □

26.

Proof. The moment generating function is

$$F(u) = E[e^{uX}] = \int_{-\infty}^{\infty} e^{ux} \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}} dx = \int_{-\infty}^{\infty} \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu-u\sigma)^2 - u^2\sigma^4 - 2\mu u\sigma^2}{2\sigma^2}} dx = e^{\frac{1}{2}u^2\sigma^2 + \mu u}.$$

So $E[X] = F'(u)|_{u=0} = \mu$, $E[X^2] = F''(u)|_{u=0} = \sigma^2 + \mu^2$, and $Var(X) = E[X^2] - (E[X])^2 = \sigma^2$. □

27.

Proof. If X is a binomial random variable with parameters (n, p), X can be written as $X = \sum_{i=1}^{n} X_i$, where $(X_i)_{i=1}^{n}$ is a sequence of i.i.d. Bernoulli random variables with parameters p. So by central limit theorem (note $E[X_i] = p$, $Var(X_i) = p(1-p)$),

$$\frac{X - np}{\sqrt{np(1-p)}} = \sqrt{n} \frac{\frac{\sum_{i=1}^{n} X_i}{n} - p}{\sqrt{p(1-p)}} \to N(0,1), \text{ as } n \to \infty.$$

This is the rationale that $P\left(\frac{X-np}{\sqrt{np(1-p)}} \le x\right) \approx \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-x^2/2} dx$ when n is large.

Remark: This provides an approximate way to compute binomial distributions while avoiding calculation of factorials. Moreover, since Poisson distribution can be approximated by binomial distribution, this exercise problem shows that, when λ is large, Poisson distribution can be approximated by $N(\lambda, \lambda)$. See Feller [2], VII.5 for details. □

28.

Proof. The Laplace transform is ($u \ge 0$)

$$F(u) = E[e^{-uX}] = \int_{-\infty}^{\infty} e^{-ux} \lambda e^{-\lambda x} dx = \frac{\lambda}{\lambda + u} = \sum_{n=1}^{\infty} \left(-\frac{u}{\lambda}\right)^n,$$

for u sufficiently small. So $E[X] = -F'(u)|_{u=0} = \frac{1}{\lambda}$, $E[X^2] = F''(u)|_{u=0} = \frac{2}{\lambda^2}$, and $Var(X) = \frac{2}{\lambda^2} - \left(\frac{1}{\lambda}\right)^2 = \frac{1}{\lambda^2}$. □

29.

[1] I didn't find a simple, elementary solution, except by calculating one-by-one the probability that A wins the match at the n-th individual game ($n = 5, 6, 7, 8, 9$).

Proof. $\frac{1}{2}$. We first give a proof without computation. By symmetry,

$$\begin{aligned} P(C \text{ is the last to leave}) &= 2P(A \text{ leaves before } B \text{ and } B \text{ leaves before } C) \\ &= 2P(B \text{ leaves before } C | A \text{ leaves before } B) P(A \text{ leaves before } B). \end{aligned}$$

Since exponential random variables are memoryless, after A leaves before B, C and B will start anew, as if they started being served simultaneously. So $P(B \text{ leaves before } C | A \text{ leaves before } B) = \frac{1}{2}$ and we can conclude $P(C \text{ is the last to leave}) = 2 \cdot \frac{1}{2} \cdot \frac{1}{2} = \frac{1}{2}$.

A direct computation is carried out as follows. Denote the service times of A, B and C by X, Y and Z, respectively, we have

$$\begin{aligned} P(C \text{ is the last to leave}) &= 1 - P(X \geq Y + Z) - P(Y \geq X + Z) \\ &= 1 - 2\int_0^\infty P(X \geq t | Y + Z = t) \lambda e^{-\lambda t} \frac{\lambda t}{1!} dt \\ &= 1 - 2\int_0^\infty e^{-\lambda t} \lambda e^{-\lambda t} \frac{\lambda t}{1!} dt \\ &= 1 - 2 \cdot \frac{1}{4} \\ &= \frac{1}{2}. \end{aligned}$$

□

30.

Proof. We note for any $t \geq 0$,

$$\begin{aligned} P(\max(X, Y) \leq t) &= P(X \leq t \text{ and } Y \leq t) = P(X \leq t) P(Y \leq t) \\ &= (1 - e^{-\lambda t})(1 - e^{-\mu t}) = 1 - e^{-\lambda t} - e^{-\mu t} + e^{-(\lambda + \mu) t}. \end{aligned}$$

So $\max(X, Y)$ is not an exponential random variable. □

31.

Proof. The probability is equal to $P(N_t = 0) = e^{-\lambda t} = e^{-0.3 \cdot 4} = 0.3012$. □

32.

Proof.

$$\begin{aligned} P(N(s) = k | N(t) = n) &= \frac{P(N(s) = k, N(t) = n)}{P(N(t) = n)} = \frac{P(N(s) = k, N(t) - N(s) = n - k)}{P(N(t) = n)} \\ &= \frac{P(N(s) = k) P(N(t - s) = n - k)}{P(N(t) = n)} = \frac{\frac{(\lambda s)^k}{k!} e^{-\lambda s} \cdot \frac{(\lambda (t-s))^{n-k}}{(n-k)!} e^{-\lambda (t-s)}}{\frac{(\lambda t)^n}{n!} e^{-\lambda t}} \\ &= \frac{s^k (t - s)^{n-k}}{t^n} \cdot \frac{n!}{k!(n-k)!} \\ &= \binom{n}{k} \left(\frac{s}{t}\right)^k \left(1 - \frac{s}{t}\right)^{n-k}. \end{aligned}$$

Remark: This gives an interesting observation: when $N(t)$ is known, the number of events by time s ($s < t$) follows a binomial distribution with parameters $(N(t), s/t)$. Equivalently, we can say given $N(t) = n$, the timings of the first n events are i.i.d. uniformly distributed over $(0, t)$. □

33.

Proof.

$$P(N(s)=k|N(t)=n)=P(N(s)-N(t)=k-n|N(t)=n)=P(N(s-t)=k-n)=\frac{(\lambda(s-t))^{k-n}}{(k-n)!}e^{-\lambda(s-t)}.$$

Remark: Intuitively, the formula should be clear: given $N(t)=n$, the conditional distribution of $N(s)$ $(s>t)$ behaves like $N(s-t)$. □

34.

Proof. The Laplace transform is $(u \geq 0)$

$$F(u)=E[e^{-uX}]=\int_0^\infty e^{-ux}\lambda e^{-\lambda x}\frac{(\lambda x)^{n-1}}{(n-1)!}dx=\frac{\lambda^n}{(\lambda+u)^n}\int_0^\infty(\lambda+u)e^{-(\lambda+u)x}\frac{((\lambda+u)x)^{n-1}}{(n-1)!}dx=\frac{\lambda^n}{(\lambda+u)^n}.$$

So $E[X]=-F'(u)|_{u=0}=\frac{n}{\lambda}$, $E[X^2]=F''(u)|_{u=0}=\frac{n(n+1)}{\lambda^2}$, and $Var(X)=E[X^2]-(E[X])^2=\frac{n}{\lambda^2}$. □

35. (a)

Proof. $E[Y|X=2]=P(Y=1|X=2)=\frac{2}{6}=\frac{1}{3}$. □

(b)

Proof. We first compute $P(Y=1)$:

$$P(Y=1)=\sum_{j=0}^{4}P(Y=1|X=j)P(X=j)=\sum_{j=0}^{4}\frac{4-j}{6}\frac{\binom{4}{j}\binom{6}{4-j}}{\binom{10}{4}}=0.4.$$

So for $j=0,1,2,3,4$,

$$P(X=j|Y=1)=\frac{P(Y=1|X=j)P(X=j)}{P(Y=1)}=\frac{\frac{4-j}{6}\binom{4}{j}\binom{6}{4-j}}{0.4\binom{10}{4}}.$$

That gives $E[X|Y=1]=\sum_{j=0}^{4}P(X=j|Y=1)\cdot j=\frac{4}{3}$.

□

(c)

Proof.

$$Var(Y|X=0)=E[Y^2|X=0]-(E[Y|X=0])^2=P(Y=1|X=0)-(P(Y=1|X=0))^2=\frac{2}{3}-\left(\frac{2}{3}\right)^2=\frac{2}{9}.$$

□

(d)

Proof.

$$Var(X|Y=1)=E[X^2|Y=1]-(E[X|Y=1])^2=\sum_{j=0}^{4}P(X=j|Y=1)\cdot j^2-\left(\frac{4}{3}\right)^2=\frac{5}{9}.$$

□

```
function ross_2_35

%ROSS_2_35  works out some routine computations for Exerciese Problem 35 of
%           Chapter 2, [1].
%
%           Reference:
%           [1] Ross: Simulation, Third Edition, Academic Press, 2002.

% Compute the probability P(Y=1)
py = 0;
denom = nchoosek(10,4);
for j = 0:4
    py = py + (4-j)/6 * nchoosek(4,j) * nchoosek(6,4-j) / denom;
end
disp(['P(Y=1) = ', num2str(py)]);

% Compute the conditional probability P(X=j|Y=1)
p = zeros(5,1);
for j = 0:4
    p(j+1) = (4-j)/6 * nchoosek(4,j) * nchoosek(6,4-j) / (denom * 0.4);
    disp(['P(X=',num2str(j),'|Y=1) = ', num2str(p(j+1))]);
end

% Compute the conditional expectation E[X|Y=1] and the conditional variance
% Var(X|Y=1)
expec = 0;
var = 0;
for j = 0:4
    expec = expec + j * p(j+1);
    var = var + j^2 * p(j+1);
end
disp(['E[X|Y=1] = ', num2str(expec)]);
disp(['Var(X|Y=1) = ', num2str(var - expec^2)]);
```

36.

Proof. $X + Y$ is a gamma random variable with parameters $(2, \lambda)$. So for $x \in [0, t]$,

$$P(X = x|X + Y = t) = \frac{P(X = x, Y = t - x)}{P(X + Y = t)} = \frac{\lambda e^{-\lambda x} \lambda e^{-\lambda(t-x)}}{\lambda e^{-\lambda t} \frac{\lambda t}{1!}} = \frac{1}{t}.$$

Remark: See page 66-67 for application. □

Chapter 3

Random Numbers

The Matlab code for the mixed congruential generator is as follows:

```
function ross_mcg(x0, a, b, m, num)

%ROSS_MCG  generates num pseudorandom numbers through the mixed congruential formula
%          x(n) = (a * x(n-1) + b) mod m.
%
%          Reference:
%          [1] Ross: Simulation, Third Edition, Academic Press, 2002.

numbers = zeros(1,num);
numbers(1) = mod(a * x0 + b, m);
for i = 2:num
    numbers(i) = mod(a * numbers(i-1) + b, m);
end
disp(numbers);
```

1.

Proof. $15, 45, 135, 105, 15, 45, 135, 105, 15, 45$. □

2.

Proof. $22, 117, 192, 167, 42, 17, 92, 67, 142, 117$. □

The Matlab code for Problem 3-9 goes as follows:

```
function ross_3_3__9

%ROSS_3_3-9  works out Exercise Problem 3 - 9 of Chapter 2, [1].
%
%            Reference:
%            [1] Ross: Simulation, Third Edition, Academic Press, 2002.

numSim = 1000000;
points = rand(numSim,1);

r3 = mean(feval('func3', points));
r4 = mean(feval('func4', points));
r5 = mean(feval('func5', points));
r7 = mean(feval('func7', points));
```

```
points1 = rand(numSim,1);
r8 = mean(feval('func8', points, points1));
r9 = mean(feval('func9', points, points1));

disp([r3, r4, r5, r7, r8, r9]);
end %ross_3_3__9

%%%%%%%%%% Nested funcitons %%%%%%%%%%%

function y = func3(x)

y = exp(exp(x));

end %func3

function y = func4(x)

y = (1-x.^2).^(1.5);

end %func4

function y = func5(x)

y = 4 * exp(16*x.^2 - 12*x + 2);

end %func5

function y = func7(x)

y = 2 * exp(-x.^2 ./ (1-x).^2) ./ (1-x).^2;

end %func7

function z = func8(x,y)

z = exp((x+y).^2);

end %func8

function z = func9(x,y)

z = (y <= x) .* exp(-x./(1-x)) .* exp(-y./(1-y)) ./ ((1-x).^2
.* (1-y).^2);

end %func9
```

3.

Proof. By change of variable formula, we have $\int_0^1 \exp\{e^x\}dx = \int_1^e e^u u^{-1} du$. Using exponential integral function $Ei(z) = -\int_{-z}^{\infty} \frac{e^{-t}}{t} dt$, the formula can be further written as $Ei(e) - Ei(1)$. Mathematica evaluates the result as 6.31656; Monte Carlo simulation gives estimates between 6.30 and 6.32. □

4.

Proof. Mathmatica gives the exact answer $\frac{3\pi}{16} \approx 0.589048622548086$; Monte Carlo simulation gives estimates between 0.5889 and 0.5891. □

5.

Proof. By the substitution $x = 4t - 2$, $\int_{-2}^{2} e^{x+x^2} dx$ is transformed to $4\int_0^1 e^{16t^2-12t+2} dt$. Mathematica evaluates this formula as 93.1628; Monte Carlo simulation gives estimates around 93.16. □

6.

Proof. The exact answer is $\frac{1}{2}$. There is no need to do Monte Carlo simulation since when the formula is transformed into an integration over $(0, 1)$, the integrand is identically 1. □

7.

Proof. $\int_{-\infty}^{\infty} e^{-x^2} dx = \int_{-\infty}^{\infty} e^{-\frac{y^2}{2}} \cdot \frac{1}{\sqrt{2}} dy = \sqrt{\pi} \approx 1.772453850905516$. To prepare for Monte Carlo simulation, we use symmetry and the substitution $x = \frac{t}{1-t}$:

$$\int_{-\infty}^{\infty} e^{-x^2} dx = 2\int_0^{\infty} e^{-x^2} dx = 2\int_0^1 \frac{e^{-\frac{t^2}{(1-t)^2}}}{(1-t)^2} dt.$$

The Monte Carlo simulation is about 1.77. □

8.

Proof. Mathematica evaluates the formula as 4.89916; Monte Carlo simulation gives estimates around 4.89. □

9.

Proof. $\int_0^{\infty}\int_0^x e^{-(x+y)} dy dx = \int_0^{\infty} e^{-x}\int_0^x e^{-y} dy dx = \int_0^{\infty} e^{-x}(1 - e^{-x}) dx = 1 - \frac{1}{2} = \frac{1}{2}$. To facilitate Monte Carlo simulation, we use the substitution $x = \frac{u}{1-u}$, $y = \frac{v}{1-v}$ and conclude

$$\int_0^{\infty}\int_0^x e^{-(x+y)} dy dx = \int_0^1\int_0^1 1_{\{v\le u\}} \frac{e^{-\left(\frac{u}{1-u}+\frac{v}{1-v}\right)}}{(1-u)^2(1-v)^2} du dv.$$

Monte Carlo simulation gives estimates around 0.5. □

10.

Proof. $Cov(X, Y) = E[XY] - E[X]E[Y]$. So $Cov(U, e^U) = \int_0^1 xe^x dx - \int_0^1 x dx \cdot \int_0^1 e^x dx = -\frac{e}{2} + \frac{3}{2} \approx 0.14086$. Monte Carlo simulation gives estimates around 0.1407. The Matlab code is the following:

```
function result = ross_3_10

%ROSS_3_10   works out Exercise Problem 10 of Chapter 2, [1].
%
%            Reference:
%            [1] Ross: Simulation, Third Edition, Academic Press, 2002.

numSim = 1000000;
U = rand(numSim,1);
result = mean(U.*exp(U)) - mean(U) * mean(exp(U));
```

□

11.

Proof. $Cov(U, \sqrt{1-U^2}) = \int_0^1 x\sqrt{1-x^2}dx - \int_0^1 xdx \cdot \int_0^1 \sqrt{1-x^2}dx = \frac{1}{3} - \frac{1}{2} \cdot \frac{\pi}{4} \approx -0.0593657$. Similarly, $Cov(U^2, \sqrt{1-U^2}) = \int_0^1 x^2\sqrt{1-x^2}dx - \int_0^1 x^2dx \cdot \int_0^1 \sqrt{1-x^2}dx = \frac{\pi}{16} - \frac{1}{3} \cdot \frac{\pi}{4} \approx -0.06545$.

Monte Carlo simulation gives estimates around -0.0594 and -0.0655, respectively. The Matlab corresponding code goes as follows:

```
function [result1, result2] = ross_3_11

%ROSS_3_11   works out Exercise Problem 11 of Chapter 2, [1].
%
%            Reference:
%            [1] Ross: Simulation, Third Edition, Academic Press, 2002.

numSim = 1000000;
U = rand(numSim,1);
result1 = mean(U.*sqrt(1-U.^2)) - mean(U)*mean(sqrt(1-U.^2));
result2 = mean(U.^2.*sqrt(1-U.^2)) - mean(U.^2)*mean(sqrt(1-U.^2));
```

□

12.

Proof. We would guess $E[N] = e$. The Matlab code for Monte Carlo simulation goes as follows:

```
function result = ross_3_12(numSim)

%ROSS_3_12   works out Exercise Problem 12 of Chapter 2, [1].
%
%            Reference:
%            [1] Ross: Simulation, Third Edition, Academic Press, 2002.

total = 0;
for i = 1:numSim
    total = total + ross_fpt_uniform;
end
result = total/numSim;

end %ross_3_12

%%%%%%%% Nested function %%%%%%%%

function counter = ross_fpt_uniform

%ROSS_FPT_UNIFORM   generates the first passage time of the sum of uniform
%                   random variables exceeding level 1.

s = 0;
counter = 0;
while (s <= 1)
    s = s + rand(1,1);
    counter = counter + 1;
end

end %ross_fpt_uniform
```

□

13.

Proof. Since $S_n = \prod_{i=1}^{n} U_i$ ($S_0 := 1$) is strictly monotone decreasing, $N = \max\{n : \prod_{i=1}^{n} U_i \geq e^{-3}\} = \min\{n : \prod_{i=1}^{n} U_i < e^{-3}\} - 1$. Monte Carlo simulation finds $E[N]$ is about 3, and $P(N = i)$ for $i = 0, 1, \cdots, 6$ are, respectively, 0.05, 0.15, 0.224, 0.224, 0.17, 0.1, 0.05. The corresponding Matlab code goes as follows:

```
function [expec, prob] = ross_3_13(numSim)

%ROSS_3_13   works out Exercise Problem 13 of Chapter 2, [1].
%
%            Reference:
%            [1] Ross: Simulation, Third Edition, Academic Press, 2002.

total = 0;
test_vec = [0, 1, 2, 3, 4, 5, 6];
count = zeros(1,7);

for i = 1:numSim
    temp = ross_fpt_loguniform;
    count = count + (test_vec == temp);
    total = total + temp;
end
expec = total/numSim; % E[N]
prob = count/numSim;  % prob(i) = P(N=i-1), i=1:7

end %ross_3_13

%%%%%%%% Nested function %%%%%%%%%

function counter = ross_fpt_loguniform

%ROSS_FPT_LOGUNIFORM   generates the first passage time of the sum of log-uniform
%                      random variables going below level -3.

s = 0;
counter = 0;
while (s >= -3)
    s = s + log(rand(1,1));
    counter = counter + 1;
end
counter = counter - 1;

end %ross_fpt_loguniform
```

Remark: By the remark on page 65, N is actually a Poisson random variable with parameter $\lambda = 3$. □

Chapter 4

Generating Discrete Random Variables

1.

Proof. The Matlab code goes as follows:

```
function result = ross_4_1(numSim)

%ROSS_4_1   works out Exercise Problem 1 of Chapter 4, [1].
%
%           We consider the possible values of the random variable
%           in decreasing order of their probabilities. Since x1
%           and x2 are not specified, we assume P(X=1)=p1, P(X=2)=p2.
%
%           Reference:
%           [1] Ross: Simulation, Third Edition, Academic Press, 2002.

result = 1 - sum(rand(1,numSim)<= 2/3)/numSim;
```

□

2.

Proof. The Matlab code goes as follows:

```
function result = ross_4_2(probs, values)

%ROSS_4_2   works out Exercise Problem 2 of Chapter 4, [1].
%
%           PROBS is a column vector that stores the probability masses,
%           and VALUES is a column vector that stores the corresponding
%           values of the random variable. We first sort PROBS in
%           descending order and VALUES is also sorted accordingly, we then
%           apply the basic procedure as described in Example 4a on page
%           46, [1].
%
%           Reference:
%           [1] Ross: Simulation, Third Edition, Academic Press, 2002.

% Sort the probability masses in descending order; the values of the random
% variable are also sorted accordingly.
```

```
sortedMat = sortrows([probs, values], -1);
cumProbs = cumsum(sortedMat(:,1));
values = sortedMat(:,2);

% Generate an incidence of the random variable.
u = rand(1,1); for i=1:numel(probs)
    if u < cumProbs(i)
        result = values(i);
        return;
    end
end
```

□

3.

Proof. Solution to Exercise Problem 2 has given the desired algorithm. The Matlab command to input is

```
ross_4_2([0.3; 0.2; 0.35; 0.15],[1; 2; 3; 4])
```

□

4.

Proof. The exact answer can be found in Feller [2], IV.4, the *matching problem*. A good approximation is that the number of hits follows Poisson distribution with parameter 1. So the expectation and the variance are both about 1. The simulation program relies on the generation of a random permutation, as described in Example 4b, page 47.

We first produce the code for generating a random permutation:

```
function randomized = ross_rperm(standard)

%ROSS_RPERM     generates a radom permutation of the entries of the input
%               vector STANDARD.
%
%               Reference:
%               [1] Ross: Simulation, Third Edition, Academic Press, 2002.


numElements = numel(standard); randomized = standard;

for k = numElements:-1:2
   index = floor(k*rand(1,1)) + 1;
   temp = randomized(k);
   randomized(k) = randomized(index);
   randomized(index) = temp;
end
```

The corresponding Matlab for the problem itself is as follows:

```
function [e, v] = ross_4_4(numSim, numCards)

%ROSS_4_4   works out Exercise Problem 4 of Chapter 4, [1]. NUMSIM
%           is the number of simulations to be carried out, and NUMCARDS
%           is the number of cards.
%
%           Reference:
%           [1] Ross: Simulation, Third Edition, Academic Press, 2002.
```

```
mysum = 0;
sqrtsum = 0;
for i = 1:numSim
    temp = sum(ross_rperm(1:numCards) == (1:numCards));
    mysum = mysum + temp;
    sqrtsum = sqrtsum + temp^2;
end
e = mysum/numSim;
v = sqrtsum/numSim - e * e;
```

□

5. (a)

Proof. The Matlab code goes as follows:

```
function randomized = ross_rperm2(standard)

%ROSS_rperm2    generates a radom permutation of the entries of the input
%               vector STANDARD. The algorithm follows Exercise Problem 5
%               of Chapter 4, [1].
%
%               Reference:
%               [1] Ross: Simulation, Third Edition, Academic Press, 2002.

numElements = numel(standard);
if (numElements == 1)
    randomized = standard;
else
    randomized = zeros(1, numElements);
    randomized(1:(numElements-1)) = ross_rperm2(standard(1:(numElements-1)));
    index = floor(rand(1,1)*numElements)+1;
    randomized(end) = randomized(index);
    randomized(index) = standard(end);
end
```

□

(b)

Proof. We denote the permutation that transforms $(1, \cdots, n)$ to $(P_1, \cdots, P_n)$ by $\begin{pmatrix} 1, \cdots, n \\ P_1, \cdots, P_n \end{pmatrix}$. For any given $(P_1, \cdots, P_n)$, if $P_{i_0} = n$, we must have

$$\begin{aligned} & P\left(\begin{pmatrix} 1, \cdots, n \\ P_1, \cdots, P_n \end{pmatrix}\right) \\ = \; & P\left(\begin{pmatrix} 1, \cdots, i_0, \cdots, n-1 \\ P_1, \cdots, P_n, \cdots, P_{n-1} \end{pmatrix}, \text{ append } n \text{ to } (P_1, \cdots, P_n, \cdots, P_{n-1}) \text{ and interchange } P_n \text{ and } n\right) \\ = \; & P\left(\begin{pmatrix} 1, \cdots, n-1 \\ P_1, \cdots, P_{n-1} \end{pmatrix}\right) \cdot \frac{1}{n} \end{aligned}$$

Clearly $P\left(\begin{pmatrix} 1 \\ 1 \end{pmatrix}\right) = 1$. If we assume $P\left(\begin{pmatrix} 1, \cdots, n-1 \\ P_1, \cdots, P_{n-1} \end{pmatrix}\right) = \frac{1}{(n-1)!}$, then the above sequence of equalities shows $P\left(\begin{pmatrix} 1, \cdots, n \\ P_1, \cdots, P_n \end{pmatrix}\right) = \frac{1}{(n-1)!} \cdot \frac{1}{n} = \frac{1}{n!}$. By induction, we can conclude that the algorithm works. □

6.

Proof. We note $\sum_{i=1}^{N} e^{\frac{i}{N}} = N \cdot \frac{\sum_{i=1}^{N} e^{\frac{i}{N}}}{N} \approx N \int_0^1 e^x dx = NE[e^U]$, where U is a uniform random variable distributed over $(0, 1)$. So with 100 uniformly distributed random numbers, we can approximate $\sum_{i=1}^{N} e^{\frac{i}{N}}$ by $N \cdot \frac{\sum_{i=1}^{100} e^{U_i}}{100}$. The approximation and comparison are carried out in the following Matlab code:

```
function [exact, approx] = ross_4_6(N, numRN)

%ROSS_4_6   works out Exercise Problem 6 of Chapter 4, [1]. In the
%           problem, N = 10,000 and numRN = 100.
%
%           Reference:
%           [1] Ross: Simulation, Third Edition, Academic Press, 2002.

exact = sum(exp((1:N)/N));
approx = N * mean(exp(rand(1,numRN)));
```

□

7.

Proof. The expected number of dice rolls is approximately 61. The Matlab code is given below:

```
function result = ross_4_7(numSim)

%ROSS_4_7   works out Exercise Problem 7 of Chapter 4, [1].
%
%           Reference:
%           [1] Ross: Simulation, Third Edition, Academic Press, 2002.

result = 0;
for i=1:numSim
    result = result + countRolls;
end
result = result/numSim;

end %ross_4_7

%%%%%%%% Nested function %%%%%%%%%%%%%

function numRolls = countRolls

%COUNTROLLS     counts the number of dice rolls that are needed for all the
%               possible outcomes to occure.

marks = ones(1,11); % marks(i)=1 iff outcome (i+1) has not occured.
numRolls = 0;
while (sum(marks) > 0)
    numRolls = numRolls + 1;
    outcome = sum( floor(6*rand(1,2)) + 1);
    marks(outcome-1) = 0;
end
end %countRolls
```

□

8.

Proof. □

9.

Proof. □

10. (a)

Proof. If X is a negative binomial random variable with parameters (r, p), it can be represented as $X = \sum_{i=1}^{r} X_i$, where $(X_i)_{i=1}^{r}$ is an i.i.d. sequence of geometric random variables with parameter p. Since a geometric random variable with parameter p can be simulated by $\text{Int}\left(\frac{\log(U)}{\log(1-p)}\right) + 1$ (U is a uniform random variable over $(0, 1)$), we can use the following program to simulate X:

```
function result = ross_4_10a(r, p)

%ROSS_4_10a works out Exercise Problem 10(a) of Chapter 4, [1].
%
%           Reference:
%           [1] Ross: Simulation, Third Edition, Academic Press, 2002.

q = 1 - p;
result = sum(floor(log(rand(1,r))/log(q)) + 1);
```

□

(b)

Proof. We note

$$\frac{p_{i+1}}{p_i} = \frac{\frac{j!}{(j+1-r)!(r-1)!}p^r(1-p)^{j+1-r}}{\frac{(j-1)!}{(j-r)!(r-1)!}p^r(1-p)^{j-r}} = \frac{j}{j+1-r}(1-p).$$

□

(c)

Proof. We note $\frac{p_{i+1}}{p_i} \geq 1$ if and only if $i \leq \frac{r-1}{p}$. So $(p_i)_{i=r}^{\infty}$ achieves its maximum at $I = Int\left(\frac{r-1}{p}\right)$if $I \geq r$. To make the algorithm efficient, we should start the search from I: if $X \leq I$, search downward from I; if $X > I$, search upward from $I + 1$. Note if we represent X as $F_X^{-1}(U)$, where F_X is the distribution function of X and U is a uniform random variable over $(0, 1)$, $X \leq I$ if and only if $U \leq F(I)$. Therefore, we have the following program:

```
function result = ross_4_10c(r, p)

%ROSS_4_10c works out Exercise Problem 10(c) of Chapter 4, [1].
%
%           Reference:
%           [1] Ross: Simulation, Third Edition, Academic Press, 2002.

u = rand(1,1);
i0 = floor((r-1)/p); % i0 is the index at which the probability achieves maximum.

if (i0 <= r)
    prob = p^r;
    cumProb = prob;
    j = r;
```

```
    while (u > cumProb)
        prob = j*(1-p)/(j+1-r) * prob;
        cumProb = cumProb + prob;
        j = j + 1;
    end
    result = j;
    return;
else
    % Generate the cumulative probability up to i0.
    prob = p^r;
    cumProb = prob;
    for j = r:(i0-1)
        prob = j*(1-p)/(j+1-r) * prob;
        cumProb = cumProb + prob;
    end % At the end of the loop, j = i0-1, prob = p(i0), and cumProb = F(i0).

    if (u > cumProb)
        % Loop for upward search. At the end of the loop, if j = n, then prob =
        % p(n+1) and cumProb = F(n+1). u satisfies F(n)< u <= F(n+1). RESULT
        % should be set as n+1 = j+1.
        while (u > cumProb)
            j = j + 1;
            prob = j*(1-p)/(j+1-r) * prob;
            cumProb = cumProb + prob;
        end
        result = j+1;
    else
        % Loop for downward search. At the end of the loop, if j = n, then prob
        % = p(n+1) and cumProb = F(n+1). u satisfies F(n+1) < u <=F(n+2).
        % RESULT should be set as n+2 = j+2.
        while (u <= cumProb && j>=(r-1))
            prob = prob * (j+1-r)/(j*(1-p));
            cumProb = cumProb - prob;
            j = j-1;
        end
        result = j+2;
    end
end
```

□

(d)

Proof.
```
function result = ross_4_10d(r, p)

%ROSS_4_10d works out Exercise Problem 10(d) of Chapter 4, [1].
%
%           Reference:
%           [1] Ross: Simulation, Third Edition, Academic Press, 2002.

result = 0;
numSuccess = 0;
while (numSuccess < r)
    if (rand(1,1) < p)
        numSuccess = numSuccess + 1;
```

```
    end
    result = result + 1;
end
```

□

11.

Proof. By symmetry of $N(0,1)$, we have

$$E[|Z|] = E[Z1_{\{Z>0\}}] + E[-Z1_{\{Z<0\}}] = 2E[Z1_{\{Z>0\}}] = 2\int_0^\infty \frac{xe^{-x^2/2}}{\sqrt{2\pi}}dx = 2\int_0^\infty \frac{e^{-u}}{\sqrt{2\pi}}du = \left(\frac{2}{\pi}\right)^2 \approx 0.798.$$

□

12.

Proof. The first method is the acceptance-rejection method. We define

$$p_i = \begin{cases} \frac{e^{-\lambda}\lambda^i/i!}{\sum_{j=0}^k e^{-\lambda}\lambda^j/j!} & 0 \le i \le k \\ 0 & \text{if } i > k \end{cases}$$

and $q_i = e^{-\lambda}\lambda^i/i!$. Define $c = \frac{1}{\sum_{j=0}^k e^{-\lambda}\lambda^j/j!}$, then $p_i/q_i \le c$. So we can first generate a Poisson random variable Y with parameter λ and then accept or reject it according to the value of an independent uniform random variable. First, we produce the Matlab code for generating a Poisson random variable:

```
function result = ross_rpoisson(lambda)

%ROSS_RPOISSON   generates a Poisson random variable with parameter lambda.
%
%                Reference:
%                [1] Ross: Simulation, Third Edition, Academic Press, 2002.

u = rand(1,1);
i0 = floor(lambda-1); % i0 is the index at which the probability achieves maximum.

if (i0 <= 0)
    prob = exp(-lambda);
    cumProb = prob;
    j = 0;
    while (u > cumProb)
        prob = lambda /(j+1) * prob;
        cumProb = cumProb + prob;
        j = j + 1;
    end
    result = j;
    return;
else
    % Generate the cumulative probability up to i0.
    prob = exp(-lambda);
    cumProb = prob;
    for j = 0:(i0-1)
        prob = lambda /(j+1) * prob;
        cumProb = cumProb + prob;
    end % At the end of the loop, j = i0-1, prob = p(i0), and cumProb = F(i0).
```

```
    if (u > cumProb)
        % Loop for upward search. At the end of the loop, if j = n, then prob =
        % p(n+1) and cumProb = F(n+1). u satisfies F(n)< u <= F(n+1). RESULT
        % should be set as n+1 = j+1.
        while (u > cumProb)
            j = j + 1;
            prob = lambda/(j+1) * prob;
            cumProb = cumProb + prob;
        end
        result = j+1;
        return;
    else
        % Loop for downward search. At the end of the loop, if j = n, then prob
        % = p(n+1) and cumProb = F(n+1). u satisfies F(n+1) < u <=F(n+2).
        % RESULT should be set as n+2 = j+2.
        while (u <= cumProb && j>=-1)
            prob = prob * (1+j)/lambda;
            cumProb = cumProb - prob;
            j = j-1;
        end
        result = j+2;
        return;
    end
end
```

Then, we note

$$\frac{p_i}{cq_i} = \begin{cases} 1 & \text{if } i \le k \\ 0 & \text{if } i > k. \end{cases}$$

So the simulation program based on the acceptance-rejection method can be written as follows:

```
function result = ross_4_12_aj(k, lambda)

%ROSS_4_12_AJ   works out Exercise Problem 12 of Chapter 4, [1], by the
%               acceptance-rejection method.
%
%               Reference:
%               [1] Ross: Simulation, Third Edition, Academic Press, 2002.

% Acceptance-rejection method
y = ross_rpoisson(lambda);
while (y > k)
    y = ross_rpoisson(lambda);
end
result = y;

end %ross_4_12_aj
```

For the second method, we note $p_i/p_{i-1} = \lambda/i$ $(i = 1, \cdots, k)$. So a Matlab program based on the inverse transform method (without any optimization of the efficiency) is as follows:

```
function result = ross_4_12(k, lambda)

%ROSS_4_12      works out Exercise Problem 12 of Chapter 4, [1].
%
```

```
%               Reference:
%               [1] Ross: Simulation, Third Edition, Academic Press, 2002.

probs = zeros(1,k+1);
probs(1) = 1;
for i = 2:(k+1)
    probs(i) = probs(i-1) * lambda / (i-1);
end
probs = probs/sum(probs);
cumProb = cumsum(probs);

u = rand(1,1);
for i = 1:(k+1)
    if (u <= cumProb(i))
        result = i-1;
        return;
    end
end

end %ross_4_12
```

□

13. (a)

Proof. We note $P(Y=k) = \frac{P(X=k)}{P(X\geq k)} = \binom{n}{k} p^k (1-p)^{n-k} \cdot \frac{1}{\alpha}$, and for $i = k, \cdots, n-1$,

$$\frac{P(Y=i+1)}{P(Y=i)} = \frac{n-i}{i+1}\frac{p}{1-p}.$$

So the Matlab code based on the inverse transform method is as follows:

```
function result = ross_4_13a(n, k, p)

%ROSS_4_13a     works out Exercise Problem 13(a) of Chapter 4, [1].
%
%               Reference:
%               [1] Ross: Simulation, Third Edition, Academic Press, 2002.

% Calculate P(X >= k) and P(X=i), i=k, ..., n.
probs = zeros(1, n-k+1);
probs(1) = nchoosek(n,k) * p^k * (1-p)^k;
for i = (k+1):n
    probs(i-k+1) = probs(i-k) * (n-i+1)/i * p/(1-p);
end
alpha = sum(probs);
probs = probs/alpha;
cumProb = cumsum(probs);

% Generate the random number via the inverse transform method.
u = rand(1,1);
j = 1;
while (u > cumProb(j))
    j = j + 1;
end
```

```
result = j;

end %ross_4_13a
```

□

(b)

Proof. We can generate Y based on the acceptance-rejection method. The solution is similar to that of Problem 12. So we omit the details. Also, we use Matlab's random generator to generate a binomial random variable, instead of implementing our own.

```
function result = ross_4_13b(n, k, p)

%ROSS_4_13b     works out Exercise Problem 13(b) of Chapter 4, [1], via the
%               acceptance-rejection method.
%
%               Reference:
%               [1] Ross: Simulation, Third Edition, Academic Press, 2002.

y = random('bino', n, p);
while (y < k)
    y = random('bino', n, p);
end
result = y;

end %ross_4_13b
```

□

(c)

Proof. Since in (b) we used the acceptance-rejection method, on average we need $1/\alpha$ binomial random variables to generate a Y. So when α is small, the algorithm in (b) is inefficient. □

14.

Proof. We note p_j can be represented as $p_j = 0.55p_j^{(1)} + 0.45p_j^{(2)}$, where $p_j^{(1)}$ is a uniform random variable over $\{5, 7, 9, 11, 13\}$ and $p_j^{(2)}$ is a uniform variable over $\{6, 8, 10, 12, 14\}$. Therefore, the Matlab code based on the composition approach is as follows:

```
function result = ross_4_14

%ROSS_4_14      works out Exercise Problem 14 of Chapter 4, [1], via the
%               composition approach.
%
%               Reference:
%               [1] Ross: Simulation, Third Edition, Academic Press, 2002.

u = rand(1,2);
if u(1)<0.55
    samples = [5, 7, 9, 11, 13];
    result = samples(floor(5*u(2)) + 1);
else
    samples = [6, 8, 10, 12, 14];
    result = samples(floor(5*u(2)) + 1);
end
```

```
end %ross_4_14
```

□

15.

Proof. We note the probability function $(p_i)_{i=1}^{10}$ can be represented as the following:

$$p_i = 0.6p_i^{(1)} + 0.35p_i^{(2)} + 0.02p_i^{(3)} + 0.01p_i^{(4)} + 0.02p_i^{(5)},$$

where $p^{(1)}$ is the uniform distribution over $\{1, \cdots, 10\}$, $p^{(2)}$ is the uniform distribution over $\{6, \cdots, 10\}$, $p^{(3)}$ is the Dirac distribution on 6, $p^{(4)}$ is the Dirac distribution on 8, and $p^{(5)}$ is the Dirac distribution on 9. So we can have the following simulation program:

```
function result = ross_4_15

%ROSS_4_15      works out Exercise Problem 15 of Chapter 4, [1], via the
%               composition approach.
%
%               Reference:
%               [1] Ross: Simulation, Third Edition, Academic Press, 2002.

u = rand(1,2);
if (u(1) <= 0.6)
    result = floor(10*u(2))+1;
elseif (u(1) <= 0.95)
    result = floor(5*u(2))+1;
elseif (u(1) <= 0.97)
    result = 6;
elseif (u(1) <= 0.98)
    result = 8;
else
    result = 9;
end

end %ross_4_15
```

□

16.

Proof. Let $p^{(1)}$ be the probability mass function that satisfies $p_j^{(1)} = \frac{1}{2^j}$ $(j = 1, 2, \cdots)$ and $p^{(2)}$ the probability mass function that satisfies $p_j^{(2)} = \frac{2^{j-1}}{3^j}$ $(j = 1, 2, \cdots)$. Then $p_j = P(X = j) = \frac{1}{2}(p_j^{(1)} + p_j^{(2)})$. This gives us a simulation program based on the composition approach:

```
function result = ross_4_16

%ROSS_4_16      works out Exercise Problem 16 of Chapter 4, [1], via the
%               composition approach.
%
%               Reference:
%               [1] Ross: Simulation, Third Edition, Academic Press, 2002.

u = rand(1,2);
if (u(1) <= 0.5)
    prob = 0.5;
```

```
    cumProb = prob;
    result = 1;
    while (u(2) > cumProb)
        prob = prob * 0.5;
        cumProb = cumProb + prob;
        result = result + 1;
    end
else
    prob = 1/3;
    cumProb = prob;
    result = 1;
    while (u(2) > cumProb)
        prob = prob * 2/3;
        cumProb = cumProb + prob;
        result = result + 1;
    end
end

end %ross_4_16
```

□

17. (a)

Proof. $\lambda_n = \frac{P(X=n)}{P(X>n-1)}$, so $1-\lambda_n = \frac{P(X>n)}{P(X>n-1)}$ and

$$(1-\lambda_1)\cdots(1-\lambda_{n-1})\lambda_n = \frac{P(X>1)}{P(X>0)}\cdots\frac{P(X>n-1)}{P(X>n-2)}\cdot\frac{P(X=n)}{P(X>n-1)} = P(X=n) = p_n.$$

□

(b)

Proof. $P(X=1) = P(U<\lambda_1) = \lambda_1 = \frac{P(X=1)}{P(X>0)} = p_1$. For $n>1$, we have by part (a)

$$P(X=n) = P(U_1 \geq \lambda_1, \cdots, U_{n-1} \geq \lambda_{n-1}, U_n < \lambda_n) = (1-\lambda_1)\cdots(1-\lambda_{n-1})\lambda_n = p_n.$$

□

(c)

Proof. If X is a geometric random variable with parameter p, then $p_n = (1-p)^{n-1}p$ and

$$\lambda = \frac{p_n}{1-\sum_{j=1}^{n-1} p_j} = \frac{(1-p)^{n-1}p}{1-\sum_{j=1}^{n-1}(1-p)^{j-1}p} = \frac{(1-p)^{n-1}p}{1-p\frac{1-(1-p)^{n-1}}{1-(1-p)}} = \frac{(1-p)^{n-1}p}{(1-p)^{n-1}} = p.$$

In this case, each iteration of step 2-3 is an independent trial which examines if the trial's outcome is a success. This essentially uses the definition of X as the number of the first trial that is a success. □

18. (a)

Proof. Y is a geometric random variable with parameter p. □

(b)

Proof. □

(c)

Proof. □

19.

Proof. The first method is to generate a random subset of size m, each element of which contributes to color type i if and only if it falls in the interval $[\sum_{j=1}^{i-1} n_j + 1, \sum_{j=1}^{i-1} n_j + n_i]$. This method involves generation of m uniform random variables (Example 4b) and m searches. It is efficient if m is small relative to n.

The second method is similar to Example 4g, where we have

$$P(X_1 = k_1) = \frac{\binom{n_1}{k_1}\binom{n - n_1}{m - k_1}}{\binom{n}{m}}$$

and

$$P(X_i = k_i | X_1 = k_1, \cdots, X_{i-1} = k_{i-1}) = \frac{\binom{n_i}{k_i}\binom{n - \sum_{j=1}^{i} n_j}{m - \sum_{j=1}^{i} k_j}}{\binom{n - \sum_{j=1}^{i-1} k_j}{m - \sum_{j=1}^{i-1} k_j}}.$$

We note each conditional distribution is a hypergeometric distribution with parameters $(n_i, n - \sum_{j=1}^{i} n_j, m - \sum_{j=1}^{i-1} k_j)$. So once the generation of a hypergeometric random variable is known, we can sequentially generate the random vector $(X_1, \cdots, X_r)$. □

Chapter 5

Generating Continuous Random Variables

1.

Proof. We note $F(x) = \int_0^x f(t)dt = \frac{e^x-1}{e-1}$. So $F^{-1}(y) = \ln[y(e-1)+1]$ and a random variable with the given density function can be simulated by $\ln[U(e-1)+1]$, where U is a uniform random variable over $(0,1)$. □

2.

Proof. For $2 \le x \le 3$, $F(x) = \int_2^x f(t)dt = \int_2^x \frac{t-2}{2}dt = (\frac{x}{2}-1)^2$. For $3 \le x \le 6$, $F(x) = \int_2^x f(t)dt = \frac{x^2}{4} - x + 1 + \int_3^x \frac{2-t/3}{2}dt = \frac{x^2}{12} - \frac{1}{2}$. So

$$F^{-1}(y) = \begin{cases} 2(\sqrt{y}+1), & 0 \le y \le \frac{1}{4} \\ \sqrt{12y+6}, & \frac{1}{4} < y \le 1. \end{cases}$$

In order to generate a random variable with the given density function, we first generate a uniform random variable U over $(0,1)$. If $U \le \frac{1}{4}$, we set $X = 2(\sqrt{U}+1)$; if $U > \frac{1}{4}$, we set $X = \sqrt{12U+6}$. □

3.

Proof. Solve the equation $y = \frac{x^2+x}{2}$, we get $F^{-1}(y) = \sqrt{2y+\frac{1}{4}} - \frac{1}{2}$. So a random variable having the given distribution function can be generated by $\sqrt{2U+\frac{1}{4}} - \frac{1}{2}$, where U is a uniform random variable over $(0,1)$. □

4.

Proof. Solve the equation $y = 1 - e^{-\alpha x^\beta}$, we get $F^{-1}(y) = \left[-\frac{\ln(1-y)}{\alpha}\right]^{1/\beta}$. So to generate a Weibull random variable, we can either use $\left[-\frac{\ln U}{\alpha}\right]^{1/\beta}$ where U is a uniform random variable over $(0,1)$, or use $Y^{1/\beta}$ where Y is an exponential random variable with rate α. □

5.

Proof. We note f is symmetric. So we first generate the absolute value of the random variable and then choose its sign according to an independent uniform random variable. More precisely, denote by X the random variable to be generated. Then the distribution function of $|X|$ is $F_{|X|}(x) = P(|X| \le x) = e^{2x} - 1$. So we generate $Y = |X|$ by $\frac{1}{2}\ln(U_1+1)$, where U_1 is uniformly distributed over $(0,1)$. Then we generate another uniform random variable U_2, which is independent of U_1, and determine X by

$$X = \begin{cases} Y, & \text{if } U \le \frac{1}{2} \\ -Y, & \text{if } U > \frac{1}{2}. \end{cases}$$

□

6.

Proof. $F(x) = \int_0^x f(t)dt = \frac{1-e^{-x}}{1-e^{-0.05}}$, $0 < x < 0.05$. So $F^{-1}(y) = -\ln[1-(1-e^{-0.05})y]$ and a random variable with given density function can be generated by $-\ln[1-(1-e^{-0.05})U]$, where U is a uniform random variable over $(0,1)$. The Matlab code to estimate $E[X|X < 0.05]$ is as follows:

```
function result = ross_5_6

%ROSS_5_6       works out Exercise Problem 6 of Chapter 5, [1].
%
%               Reference:
%               [1] Ross: Simulation, Third Edition, Academic Press, 2002.

numSim = 100;
result = mean(-log(1 - rand(1,numSim)*(1-exp(-0.05))));

end %ross_5_6
```

To calculate the exact value of $E[X|X < 0.05]$, we note

$$E[X|X < 0.05] = \int_0^{0.05} \frac{xe^{-x}}{1-e^{-0.05}}dx = 0.0248.$$

□

7.

Proof. Use a random variable with probability mass function $\{p_1, \cdots, p_n\}$ to determine which of the n distribution functions $\{F_1, \cdots, F_n\}$ will be used for the generation of the random variable, then generate the random variable according to the chosen distribution. □

8. (a)

Proof. If we define $F_1(x) = x$, $F_2(x) = x^3$, and $F_3(x) = x^5$ $(0 \le x \le 1)$, then $F(x) = \frac{1}{3}(F_1(x)+F_2(x)+F_3(x))$. The corresponding Matlab code is as follows:

```
function result = ross_5_8a

%ROSS_5_8a      works out Exercise Problem 8(a) of Chapter 5, [1].
%
%               Reference:
%               [1] Ross: Simulation, Third Edition, Academic Press, 2002.

u = rand(1,2);
if (u(1)<=1/3)
    result = u(2);
    return;
elseif (u(2)<=2/3)
    result = u(2)^(1/3);
    return;
else
    result = u(2)^(1/5);
    return;
end

end %ross_5_8a
```

□

(b)

Proof. It is easy to see the density function corresponding to $F(x)$ is

$$f(x) = \begin{cases} \frac{2}{3}(1+e^{-2x}), & \text{if } 0 < x < 1 \\ \frac{2}{3}e^{-2x}, & \text{if } 1 < x < \infty. \end{cases}$$

If we define $p_1 = \frac{2}{3}$, $p_2 = \frac{1-e^{-2}}{3}$, $p_3 = \frac{e^{-2}}{3}$, $f_1(x) = 1_{(0,,1)}(x)$, $f_2(x) = \frac{2e^{-2x}}{1-e^{-2}}1_{(0,1)}$, and $f_3(x) = 2e^{-2(x-1)}1_{(1,\infty)}(x)$, then $f(x) = p_1f_1(x) + p_2f_2(x) + p_3f_3(x)$. The corresponding distribution functions are $F_1(x) = x1_{(0,1]}(x) + 1_{(1,\infty)}$, $F_2(x) = \frac{1-e^{-2x}}{1-e^{-2}}1_{(0,1]}(x) + 1_{(1,\infty)}$, and $F_3(x) = (1-e^{-2(x-1)})1_{(1,\infty)}(x)$. The Matlab code is as follows:

```
function result = ross_5_8b

%ROSS_5_8b      works out Exercise Problem 8(b) of Chapter 5, [1].
%
%               Reference:
%               [1] Ross: Simulation, Third Edition, Academic Press, 2002.

u = rand(1,2);
if (u(1) <= 2/3)
    result = u(2);
    return;
elseif (u(1) < (3-exp(-2))/3)
    result = -0.5 * log(1 - u(2) * (1-exp(-2)));
    return;
else
    result = 1 - 0.5 * log(1-u(2));
    return;
end

end %ross_5_8b
```

□

(c)

Proof. The Matlab code is as follows:

```
function result = ross_5_8c(alpha)

%ROSS_5_8c      works out Exercise Problem 8(c) of Chapter 5, [1].
%
%               Reference:
%               [1] Ross: Simulation, Third Edition, Academic Press, 2002.

u = rand(1,2);
n = numel(alpha);
cumAlpha = cumsum(alpha);
for i=1:n
    if (u(1) <= cumAlpha(i))
        result = (u(2)*(1+i))^(1/(1+i));
        return;
```

```
    end
end

end %ross_5_8c
```

□

9.

Proof. Following the hint, we design the algorithm as follows:

Step 1: Generate an exponential random variable Y with rate 1.

Step 2: Based on the value of Y, say y, generate X according to the distribution function $F(x) = x^y 1_{(0,1]}(x) + 1_{(1,\infty)}(x)$.

It is easy to see the random variable X thus generated has the desired distribution function, since

$$P(X \leq x) = \int_0^\infty P(X \leq x | Y = y) e^{-y} dy = \int_0^\infty x^y e^{-y} dy.$$

The corresponding Matlab code is as follows:

```
function result = ross_5_9

%ROSS_5_9       works out Exercise Problem 9 of Chapter 5, [1].
%
%               Reference:
%               [1] Ross: Simulation, Third Edition, Academic Press, 2002.

y = random('exp',1);
result = (rand(1,1))^(1/y);

end %ross_5_9
```

□

10.

Proof. Denote by N the number of claims in the next month, and let $(X_i)_{i=1}^\infty$ be an i.i.d. sequence of exponential random variables with mean 800. Then N is a binomial random variable with parameter $(1000, 0.05)$. The sum S of claims made in the next month can be represented as $S = \sum_{i=1}^N X_i$. The algorithm for the simulation program can therefore be designed as follows:

Step 1: Set $C = 0$.

Step 2: Generate a binomial random variable N with parameters $(1000, 0.05)$.

Step 3: Assume $N = n$, generate n exponential random variables with mean 800 and add them up.

Step 4: If the sum obtained in Step 3 is greater than \$50,000, count 1; otherwise, count 0. Add this counting result to C.

Step 5: Go back to Step 2 until the desired number of simulation has been achieved.

The corresponding Matlab code is as follows:

```
function result = ross_5_10(numSim)

%ROSS_5_10      works out Exercise Problem 10 of Chapter 5, [1].
%
%               Reference:
%               [1] Ross: Simulation, Third Edition, Academic Press, 2002.

count = 0;
```

```
for i=1:numSim
    n = random('bino',1000,0.05);
    count = count + ( sum(random('exp',800, 1, n)) > 50000);
end
result = count/numSim;

end %ross_5_10
```

□

11.

Proof. Following the method outlined at the end of §5.1, we have the following Matlab code:

```
function result = ross_5_11(mu, k)

%ROSS_5_11      works out Exercise Problem 11 of Chapter 5, [1].
%
%               ROSS_5_11 generates a set of K i.i.d. exponential random
%               variables with mean MU.
%
%               Reference:
%               [1] Ross: Simulation, Third Edition, Academic Press, 2002.

if (k==1)
    result = random('exp', mu, 1, 1);
    return;
else
    t = -log( prod(rand(1,k)) );
    U = rand(1,k-1);
    result = diff([0, t * sort(U,'ascend'), t]);
end

end %ross_5_11
```

□

12.

Proof. We can take $X = \max\{X_1, \cdots, X_n\}$ for part (a) and $X = \min\{X_1, \cdots, X_n\}$ for part (b). □

Chapter 6

The Discrete Event Simulation Approach

Chapter 7

Statistical Analysis of Simulated Data

Chapter 8

Variance Reduction Techniques

Chapter 9

Statistical Validation Techniques

Chapter 10

Markov Chain Monte Carlo Methods

Chapter 11

Some Additional Topics

Bibliography

[1] R. Durrett. *Probability: Theory and Examples*, 2nd ed. Duxbury Press, 1995.

[2] W. Feller. *An Introduction to Probability Theory and Its Applications*, 3rd ed. Wiley, New York, 1968.