

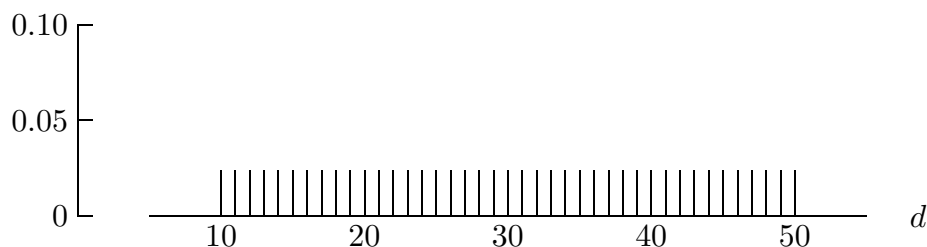
The purpose of this section is to demonstrate several discrete random variable applications using the capabilities provided by the discrete random variate generators in the library `rvgs` and the pdf, cdf, and idf functions in the library `rvms`. We begin by considering alternatives to the inventory demand models used in programs `sis2` and `sis3`.

### 6.3.1 ALTERNATIVE INVENTORY DEMAND MODELS

**Example 6.3.1** The inventory demand model in program `sis2` is that the demand per time interval is generated as an *Equilikely*(10, 50) random variate  $d$ . In this case, the mean is 30, the standard deviation is  $\sqrt{140} \cong 11.8$ , and the demand pdf is flat, as illustrated in Figure 6.3.1.

**Figure 6.3.1.**

*Equilikely*(10, 50)  
demand  
model  $f(d)$   
pdf.



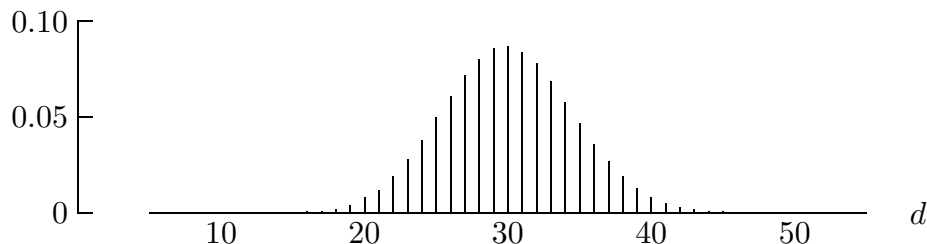
As discussed in Example 3.1.5, this is not a very realistic model.\* Therefore, we consider alternative models, one of which is that there are, say, 100 instances per time interval when a demand for 1 unit *may* occur. For each of these instances the probability that the demand *will* occur is, say, 0.3 independent of what happens at the other demand instances. The inventory demand per time interval is then the sum of 100 independent *Bernoulli*(0.3) random variates or, equivalently, it is a *Binomial*(100, 0.3) random variate. In this case the function `GetDemand` in the program `sis2` should be

```
long GetDemand(void)
{
    return (Binomial(100, 0.3));
}
```

The resulting random demand per time interval will have a mean of 30, a standard deviation of  $\sqrt{21} \cong 4.6$ , and a pdf approximately symmetric about the mean, as illustrated in Figure 6.3.2.

**Figure 6.3.2.**

*Binomial*(100, 0.3)  
demand  
model  $f(d)$   
pdf.



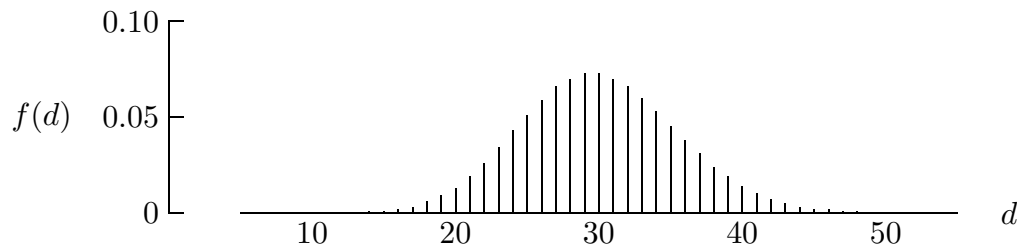

---

\* Discussion of matching models to data collected from the system of interest is delayed until Chapter 9.

**Example 6.3.2** If we believe that the  $Binomial(100, 0.3)$  model in the previous example may be close to reality, then a  $Poisson(30)$  model should also be considered. For this model, the random variate returned by the function `GetDemand` in program `sis2` would be

```
return (Poisson(30.0));
```

The resulting random variate demand per time interval will have a mean of 30, a standard deviation of  $\sqrt{30} \cong 5.5$ , and the pdf illustrated in Figure 6.3.3. Although similar to the  $Binomial(100, 0.3)$  pdf in Example 6.3.1, the  $Poisson(30)$  pdf has slightly “heavier” tails, consistent with the larger standard deviation.



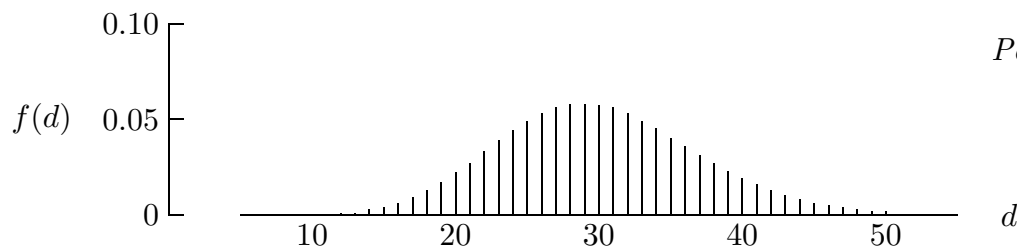
**Figure 6.3.3.**  
*Poisson(30)*  
demand  
model  
pdf.

For reasons discussed in Section 7.3, a traditional model for inventory systems is what was used in Example 6.3.2 — the discrete demand is a  $Poisson(\lambda)$  random variable with the parameter  $\lambda$  matched to the expected amount of demand per time interval (the demand rate). Indeed, this is the inventory demand model used in program `sis3` with  $\lambda = 30$ .

**Example 6.3.3** Yet another potential model for the inventory demand is that there are, say, 50 instances per time interval when a  $Geometric(p)$  inventory demand will occur with  $p$  equal to, say, 0.375 and the amount of demand that actually occurs at each of these instances is independent of the amount of demand that occurs at the other instances. The demand per time interval is then the sum of 50 independent  $Geometric(0.375)$  random variates or, equivalently it is a  $Pascal(50, 0.375)$  random variate. The random variate returned by the function `GetDemand` in program `sis2` would then be

```
return (Pascal(50, 0.375));
```

producing a random variate demand per time interval with a mean of 30, a standard deviation of  $\sqrt{48} \cong 6.9$ , and the pdf illustrated in Figure 6.3.4.



**Figure 6.3.4.**  
*Pascal(50, 0.375)*  
demand  
model  
pdf.

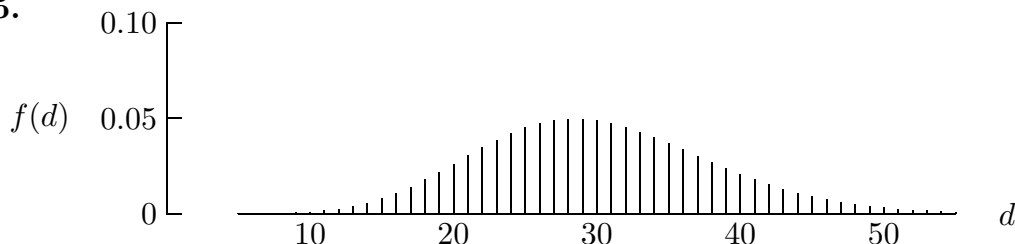
Consistent with the increase in the standard deviation from 5.5 to 6.9, this pdf has slightly heavier tails than the  $Poisson(30)$  pdf.

**Example 6.3.4** As an extension of the previous example, one might argue that the inventory demand model should also allow for the number of demand instances per time interval to be an independent discrete random variable as well, say *Poisson*(50). The function `GetDemand` in program `sis2` would then be\*

```
long GetDemand(void)
{
    long instances = Poisson(50.0);      /* must truncate to avoid 0 */
    return (Pascal(instances, 0.375));
}
```

With this extension, the mean of the resulting random variate demand per time interval will remain 30. As illustrated in Figure 6.3.5, the pdf will become further dispersed about the mean, consistent with an increase of the standard deviation to  $\sqrt{66} \cong 8.1$ .

**Figure 6.3.5.**  
Compound  
demand  
model  
pdf.



To determine the pdf of the “compound” (or “stochastic parameters”) random variable corresponding to the random variates generated by the function `GetDemand` in Example 6.3.4, let the discrete random variables  $D$  and  $I$  denote the demand *amount* and number of demand *instances* per time interval, respectively. From the law of total probability it follows that

$$f(d) = \Pr(D = d) = \sum_{i=0}^{\infty} \Pr(I = i) \Pr(D = d \mid I = i) \quad d = 0, 1, 2, \dots$$

For any value of  $d$  the probability  $f(d)$  can be evaluated by using the pdf capability in the library `rvms`. To do so, however, the infinite sum over possible demand instances must be *truncated* to a finite range, say  $0 < a \leq i \leq b$ . The details of how to determine  $a$  and  $b$  are discussed later in this section. Provided  $a$  and  $b$  are selected appropriately, the following algorithm can then be used to compute  $f(d)$

```
double sum = 0.0;
for (i = a; i <= b; i++)          /* use the library rvms */
    sum += pdfPoisson(50.0, i) * pdfPascal(i, 0.375, d);
return sum;                       /* sum is f(d) */
```

These compound random variables are of interest in a branch of statistics known as Bayesian statistics.

---

\* There is a potential problem with this function — see Exercise 6.3.7.

Any of the inventory demand models in the previous examples can be used in program `sis2` as a replacement for the *Equilikely*( $a, b$ ) model. In most applications this will result in a model that more accurately reflects the system. Moreover, with some minor modification, a more accurate inventory demand model can be used in program `sis3`. The result is program `sis4`.

### Program `sis4`

Program `sis4` is based on the next-event simulation program `sis3` but with a more realistic inventory demand model that allows for a random amount of demand at each demand instance. That is, like program `sis3` demand instances are assumed to occur at random throughout the period of operation with an average rate of  $\lambda$  instances per time interval so that the inter-demand time is an *Exponential*( $1/\lambda$ ) random variate. Unlike the model on which program `sis3` is based, however, these demand instances correspond to times where a demand *may* occur. Whether or not a demand actually occurs at these demand instances is random with probability  $p$ . Moreover, to allow for the possibility of more than one unit of inventory demand at a demand instance, the demand amount is assumed to be a *Geometric*( $p$ ) random variate. Because the expected value of a *Geometric*( $p$ ) random variable is  $p/(1-p)$  and the expected number of demand instances per time interval is  $\lambda$ , the expected demand per time interval is  $\lambda p/(1-p)$ .

**Example 6.3.5** In terms of the automobile dealership example considered previously, the inventory demand model on which program `sis4` is based corresponds to an average of  $\lambda$  customers per week that visit the dealership with the *potential* to buy one or more automobiles. Each customer will, independently, not buy an automobile with probability  $1-p$ , or they will buy one automobile with probability  $(1-p)p$ , or they will buy two automobiles with probability  $(1-p)p^2$ , or three with probability  $(1-p)p^3$ , etc. The parameter values used in program `sis4` are  $\lambda = 120.0$  and  $p = 0.2$ , which correspond to an expected value of 30.0 automobiles purchased per week. For future reference, note that

$$30.0 = \frac{\lambda p}{1-p} = \lambda \sum_{x=0}^{\infty} x(1-p)p^x = \underbrace{\lambda(1-p)p}_{19.200} + \underbrace{2\lambda(1-p)p^2}_{7.680} + \underbrace{3\lambda(1-p)p^3}_{2.304} + \dots$$

Therefore, on average, of the 120 customers that visit per week

- $\lambda(1-p) = 96.0$  do not buy anything;
- $\lambda(1-p)p = 19.200$  buy one automobile;
- $\lambda(1-p)p^2 = 3.840$  buy two automobiles;
- $\lambda(1-p)p^3 = 0.768$  buy three automobiles;
- etc.

In the remainder of this section we will deal with the issue of how to truncate a discrete random variable. By using truncation the demand model can be made more realistic by limiting the number of automobiles a customer buys to 0, 1 or 2 (see Exercise 6.3.2).

### 6.3.2 TRUNCATION

Let  $X$  be a discrete random variable with possible values  $\mathcal{X} = \{x \mid x = 0, 1, 2, \dots\}$  and cdf  $F(x) = \Pr(X \leq x)$ . A modeler might be interested in formulating a pdf that effectively restricts the possible values to a finite range of integers  $a \leq x \leq b$  with  $a \geq 0$  and  $b < \infty$ .<sup>\*</sup> If  $a > 0$  then the probability of  $X$  being strictly less than  $a$  is

$$\alpha = \Pr(X < a) = \Pr(X \leq a - 1) = F(a - 1).$$

Similarly, the probability of  $X$  being strictly greater than  $b$  is

$$\beta = \Pr(X > b) = 1 - \Pr(X \leq b) = 1 - F(b).$$

In general, then

$$\Pr(a \leq X \leq b) = \Pr(X \leq b) - \Pr(X < a) = F(b) - F(a - 1)$$

so that a value of  $X$  outside the range  $a \leq X \leq b$  is essentially impossible if and only if  $F(b) \cong 1.0$  and  $F(a - 1) \cong 0.0$ . There are two cases to consider.

- If  $a$  and  $b$  are specified then the cdf of  $X$  can be used to determine the left-tail and right-tail probabilities

$$\alpha = \Pr(X < a) = F(a - 1) \quad \text{and} \quad \beta = \Pr(X > b) = 1 - F(b)$$

respectively. The cdf transformation from possible values to probabilities is exact.

- If instead the (positive) left-tail and right-tail probabilities  $\alpha$  and  $\beta$  are specified, then the idf of  $X$  can be used to determine the possible values

$$a = F^*(\alpha) \quad \text{and} \quad b = F^*(1 - \beta)$$

respectively. Because  $X$  is a discrete random variable, this idf transformation from probabilities to possible values is not exact. Instead  $a$  and  $b$  only provide *bounds* in the sense that  $\Pr(X < a) \leq \alpha$  and  $\Pr(X > b) < \beta$ .

**Example 6.3.6** For the *Poisson*(50) random variable  $I$  in Example 6.3.4 it was necessary to determine  $a, b$  so that  $\Pr(a \leq I \leq b) \cong 1.0$  with negligible error. By experimentation it was determined that, in this case, “negligible error” could be interpreted as  $\alpha = \beta = 10^{-6}$ . The idf capability in the random variate models library `rvms` was then used to compute

```
a = idfPoisson(50.0, alpha);           /* alpha = 10-6 */
b = idfPoisson(50.0, 1.0 - beta);     /* beta = 10-6 */
```

to produce the results  $a = 20, b = 87$ . Consistent with the bounds produced by the  $(\alpha, \beta)$  to  $(a, b)$  conversion,  $\Pr(I < 20) = \text{cdfPoisson}(50.0, 19) \cong 0.48 \times 10^{-6} < \alpha$  and  $\Pr(I > 87) = 1.0 - \text{cdfPoisson}(50.0, 87) \cong 0.75 \times 10^{-6} < \beta$ .

---

<sup>\*</sup> In the case of the automobile dealership example,  $a = 0$  and  $b = 45$  implies that there is no left-hand truncation and no week contains more than 45 customer demands.

To facilitate the evaluation of the compound pdf values in Example 6.3.4, the possible values of a  $Poisson(50)$  random variable were truncated to the integers 20 through 87 inclusive. As demonstrated in Example 6.3.6, this truncation is so slight as to be of no practical significance. That is, because a  $Poisson(50)$  pdf is effectively zero for all integers less than 20 or greater than 87, the truncated random variable and the un-truncated random variable are essentially identically distributed. In some simulation applications truncation, is more significant. There are two reasons for this.

- *Efficiency* — when discrete random variates are generated using inversion and the idf is not available as a simple algebraic expression, then a cdf search technique like Algorithm 6.2.2 must be used. To facilitate this search, cdf values are usually stored in an array. Because the range of possible values defines the size of the array, traditional computer science space/time considerations dictate that the range of possible values should be as small as possible (assuming that the model remains realistic).
- *Realism* — some discrete random variates can take on arbitrarily large values (at least in theory). If you have created a discrete random variable model with a mean of 30 and a standard deviation of 10 do you really want your random variate generator to (surprise!) return a value of 100?

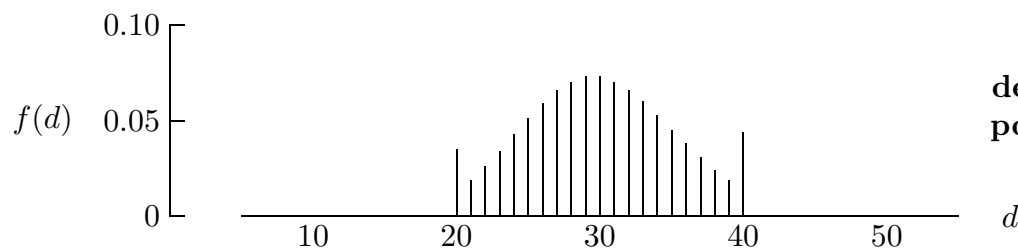
When it is significant, truncation must be done correctly because the result is a *new* random variable. The four examples that follow are illustrations.

### Incorrect Truncation

**Example 6.3.7** As in Example 6.3.2, suppose we want to use a  $Poisson(30)$  demand model in program `sis2` but, to avoid extreme values, we want to truncate the demand to the range  $20 \leq d \leq 40$ . In this case, we might be tempted to generate the demand with

```
d = Poisson(30.0);
if (d < 20)
    d = 20;
if (d > 40)
    d = 40;
return d;
```

Because the pdf values below 20 and above 40 “fold” back to 20 and 40 respectively, as the resulting pdf spikes at  $d = 20$  and  $d = 40$  shown in Figure 6.3.6 indicate this is *not* a correct truncation for most applications.



**Figure 6.3.6.**  
*Poisson(30)*  
demand model  
pdf incorrectly  
truncated.

**Truncation by cdf Modification**

**Example 6.3.8** As in Example 6.3.7, we want to generate  $Poisson(30)$  demands truncated to the range  $20 \leq d \leq 40$ . This time, however, we want to do it correctly. Before truncation the  $Poisson(30)$  pdf is

$$f(d) = \frac{30^d \exp(-30)}{d!} \quad d = 0, 1, 2, \dots$$

with

$$\Pr(20 \leq D \leq 40) = F(40) - F(19) = \sum_{d=20}^{40} f(d) \cong 0.945817.$$

The truncation correction is to compute a new truncated random variable  $D_t$  with pdf  $f_t(d)$ . This is done by increasing the corresponding value of  $f(d)$  via division by the factor  $F(40) - F(19)$  as

$$f_t(d) = \frac{f(d)}{F(40) - F(19)} \quad d = 20, 21, \dots, 40.$$

The corresponding truncated cdf is

$$F_t(d) = \sum_{t=20}^d f_t(t) = \frac{F(d) - F(19)}{F(40) - F(19)} \quad d = 20, 21, \dots, 40.$$

A random variate correctly truncated to the range  $20 \leq d \leq 40$  can now be generated by inversion using the truncated cdf  $F_t(\cdot)$  and Algorithm 6.2.2 as follows

```

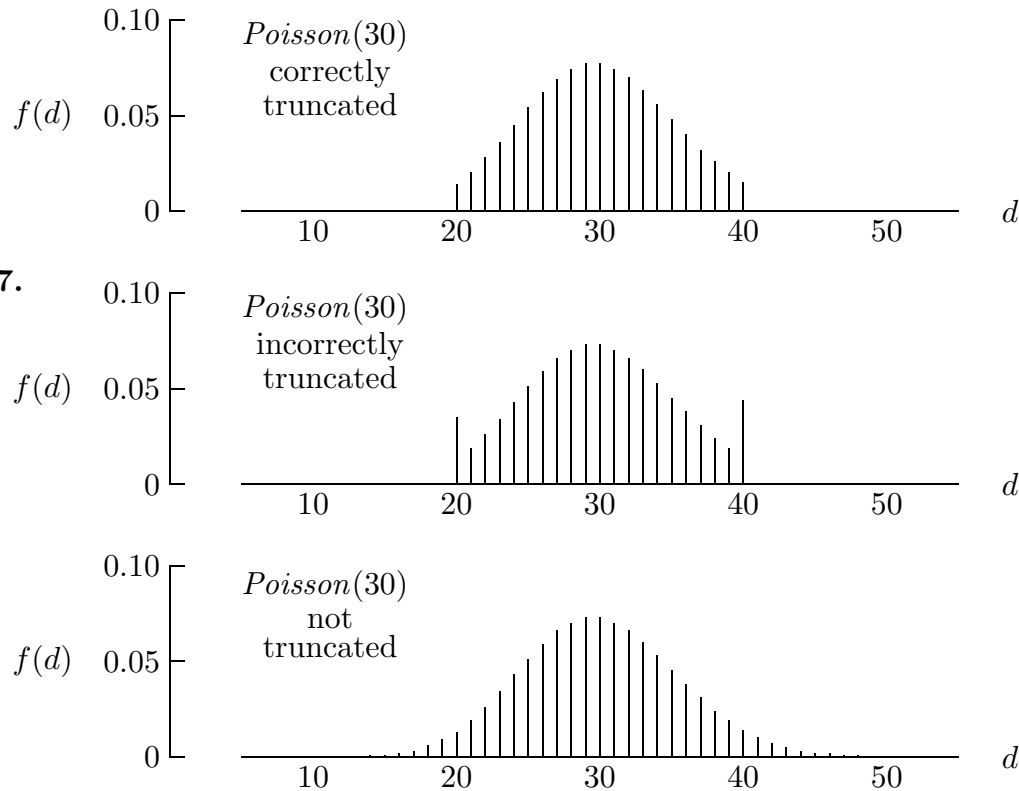
u = Random();
d = 30;
if (F_t(d) <= u)
    while (F_t(d) <= u)
        d++;
else if (F_t(20) <= u)
    while (F_t(d-1) > u)
        d--;
else
    d = 20;
return d;

```

Compared to the mean and standard deviation of  $D$ , which are  $\mu = 30.0$  and  $\sigma = \sqrt{30} \cong 5.477$ , the mean and standard deviation of  $D_t$  are

$$\mu_t = \sum_{d=20}^{40} d f_t(d) \cong 29.841 \quad \text{and} \quad \sigma_t = \sqrt{\sum_{d=20}^{40} (d - \mu_t)^2 f_t(d)} \cong 4.720.$$

The pdf of the correctly truncated  $Poisson(30)$  random variable in Example 6.3.8 is illustrated in Figure 6.3.7. For reference, the incorrectly truncated pdf from Example 6.3.7 and the un-truncated pdf from Example 6.3.2 are also shown.



**Figure 6.3.7.**  
Various  
 $Poisson(30)$   
demand  
model  
pdfs.

In general, if it is desired to truncate the integer-valued discrete random variable  $X$  to the set of possible values  $\mathcal{X}_t = \{a, a + 1, \dots, b\} \subset \mathcal{X}$  thereby forming the new discrete random variable  $X_t$ , then the pdf and cdf of  $X_t$  are defined as

$$f_t(x) = \frac{f(x)}{F(b) - F(a - 1)} \quad x \in \mathcal{X}_t$$

and

$$F_t(x) = \frac{F(x) - F(a - 1)}{F(b) - F(a - 1)} \quad x \in \mathcal{X}_t$$

where  $f(\cdot)$  and  $F(\cdot)$  are the pdf and cdf of  $X$ . Random values of  $X_t$  can then be generated using inversion and Algorithm 6.2.2 with  $F_t(\cdot)$ , as in Example 6.3.8. [The equations for  $f_t(x)$  and  $F_t(x)$  presume that  $a - 1$  is a possible value of  $X$ .]

There may be *rare* occasions where “incorrect truncation” is appropriate in a discrete-event simulation model. Decisions concerning the style of truncation are driven by the fit between the model and the real-world system.

As an alternative to truncation by cdf modification, we can use truncation by *constrained inversion*, as illustrated next. Provided the idf of  $X$  can be easily applied, this is the truncation approach of choice.

### Truncation by Constrained Inversion

If  $F^*(\cdot)$  is the idf of  $X$  then the following algorithm can be used to generate the random variate  $X_t$  correctly truncated to the range  $a \leq x \leq b$

```

 $\alpha = F(a - 1);$           /* assumes that  $a - 1$  is a possible value of  $X$  */
 $\beta = 1.0 - F(b);$ 
 $u = \text{Uniform}(\alpha, 1.0 - \beta);$ 
 $x = F^*(u);$ 
return  $x;$ 

```

The key here is that  $u$  is *constrained* to a subrange  $(\alpha, 1 - \beta) \subset (0, 1)$  in such a way that correct truncation is automatically enforced prior to the idf inversion. This is another illustration of the elegance of random variate generation by inversion.

**Example 6.3.9** The cdf and idf capabilities in the library `rvms` can be used to generate a  $Poisson(30)$  random demand, correctly truncated to the range  $20 \leq d \leq 40$ , as illustrated

```

 $\alpha = \text{cdfPoisson}(30.0, 19);$           /* set-up */
 $\beta = 1.0 - \text{cdfPoisson}(30.0, 40);$     /* set-up */
 $u = \text{Uniform}(\alpha, 1.0 - \beta);$ 
 $d = \text{idfPoisson}(30.0, u);$ 
return  $d;$ 

```

This algorithm should be implemented so that  $\alpha$  and  $\beta$  are `static` variables that are computed once only.

### Truncation by Acceptance-Rejection

**Example 6.3.10** As an alternative to the techniques in Example 6.3.8 and 6.3.9, correct truncation can also be achieved, at the potential expense of some extra calls to the function `Poisson(30)`, by using *acceptance-rejection* as follows

```

 $d = \text{Poisson}(30.0);$ 
while (( $d < 20$ ) or ( $d > 40$ ))
     $d = \text{Poisson}(30.0);$ 
return  $d;$ 

```

Although easily remembered and easily programmed, acceptance-rejection is not synchronized or monotone even if the un-truncated generator has these properties. Generally, the cdf modification technique in Example 6.3.8 or the constrained inversion technique in Example 6.3.9 is preferable.

## 6.3.3 EXERCISES

**Exercise 6.3.1** (a) Suppose you wish to use inversion to generate a  $Binomial(100, 0.1)$  random variate  $X$  truncated to the subrange  $x = 4, 5, \dots, 16$ . How would you do it? Work through the details. (b) What is the value of the mean and standard deviation of the resulting truncated random variate?

**Exercise 6.3.2** The function `GetDemand` in program `sis4` can return demand amounts outside the range 0, 1, 2. (a) What is the largest demand amount that this function can return? In some applications, integers outside the range 0, 1, 2 may not be meaningful, no matter how unlikely. (b) Modify `GetDemand` so that the value returned is correctly truncated to the range 0, 1, 2. Do not use acceptance-rejection. (c) With truncation, what is the resulting average demand per time interval and how does that compare to the average with no truncation?

**Exercise 6.3.3** Prove that the “truncation by constrained inversion” algorithm is correct. Also, draw a figure that illustrates the geometry behind the “truncation by constrained inversion” algorithm.

**Exercise 6.3.4<sup>a</sup>** Prove that the mean of the compound demand in Example 6.3.4 is 30 and that the variance is 66.

**Exercise 6.3.5<sup>a</sup>** (a) Do Exercise 6.2.10. (b) In addition, prove that the expected value of  $X$  is infinitely large (i.e.,  $E[X] = \infty$ .) (c) Comment on the potential paradox that by using inversion just *one* call to `Random` is somehow equivalent to a direct Monte Carlo simulation of this random experiment that requires, on average, an *infinite* number of calls to `Random`. *Hint*: the pdf of  $X$  is

$$f(x) = \frac{1}{x(x+1)} = \frac{1}{x} - \frac{1}{x+1} \quad x = 1, 2, 3, \dots$$

**Exercise 6.3.6<sup>a</sup>** (a) Implement the function `GetDemand` in Example 6.3.4 in such a way that you can use Monte Carlo simulation to estimate the expected number of calls to `Random` per call to `GetDemand`. (b) Construct an alternative version of `GetDemand` that requires exactly *one* call to `Random` per call to `GetDemand` and yet is effectively equivalent to the original function in the sense that the alternative function produces a random variate whose pdf matches the pdf illustrated in Example 6.3.4. (c) Provide convincing numerical evidence of correctness.

**Exercise 6.3.7** (a) Is it necessary to modify the function `GetDemand` in Example 6.3.4 so that `instances` must be positive? (b) If so, why and how would you do it? If not necessary, why?