Discrete-Event Simulation:

A First Course

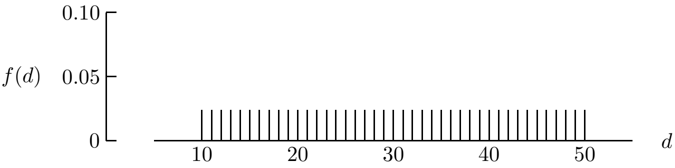
Section 6.3: Discrete Random Variable Applications

0-13-142917-5

Section 6.3: Discrete Random Variable Applications

Example 6.3.1: The inventory demand model in program `sis2`

- The demand per time interval is an *Equilikely*(10,50) random variate
- $\mu = 30$, $\sigma = \sqrt{140} \cong 11.8$, and the demand pdf is flat



- This model is not very realistic (see Chapter 9)

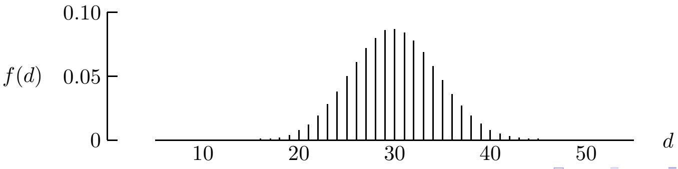
Alternative Inventory Demand Model

- Consider a *Binomial*(100,0.3) model
  - 100 instances per time interval when demand for 1 unit may occur
  - The probability of demand is 0.3 per instance (independently)
  - The function `GetDemand` in `sis2` becomes:

Modified GetDemand Method

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

- $\mu = 30$, $\sigma = \sqrt{21} \cong 4.6$ and the pdf is:



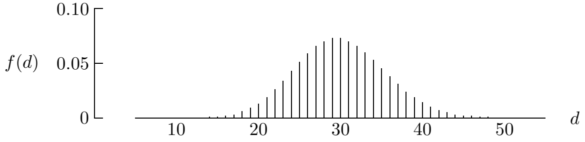
Example 6.3.2: A Poisson(30) Model

- Recall that *Binomial*$(n, p) \approx$ *Poisson*(np) for large n
- If *Binomial*(100, 0.3) is realistic, should also consider *Poisson*(30)
- The function `GetDemand` in program `sis2` would be

Modified GetDemand Method

```
long GetDemand(void) {
    return (Poisson(30.0));
}
```

- $\mu = 30$, $\sigma = \sqrt{30} \cong 5.5$ and the pdf has slightly "heavier" tails



- *Poisson*(λ) is the inventory demand model used in `sis3` with $\lambda = 30$

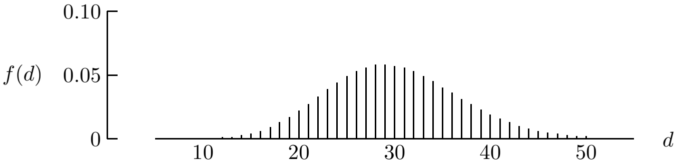
Example 6.3.3: A Pascal(50,0.375) Model

- 50 instances per time interval
- The demand per instance is *Geometric(p)* with $p = 0.375$
- The function `GetDemand` in program `sis2` would be

Modified GetDemand Method

```
long GetDemand(void) {
    return return (Pascal(50,0.375));
}
```

- $\mu = 30$, $\sigma = \sqrt{48} \cong 6.9$ and the pdf has heavier tails than the *Poisson*(30) pdf



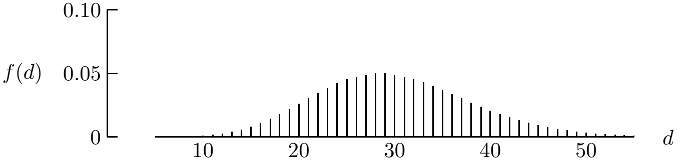
Example 6.3.4

- The number of demand instances per time interval is *Poisson*(50)
- The demand per instance is *Geometric(p)* with $p = 0.375$

Modified GetDemand Method

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

- $\mu = 30$, $\sigma = \sqrt{66} \cong 8.1$ and the pdf has heavier tails



The pdf in Example 6.3.4

- Define random variables

  D: the demand *amount*

  I : the number of demand *instances* per time interval

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

- To compute $f(d)$, truncate infinite sum: $0 < a \leq i \leq b$

Computing $f(d)$

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

Program sis4

- Based on `sis3` but with a more realistic inventory demand model
- The inter-demand time is an *Exponential*$(1/\lambda)$ random variate
- Whether or not a demand occurs at demand instances is random with probability p
- To allow for the possibility of more than 1 unit of demand, the demand amount is a *Geometric*(p) random variate
- Expected demand per time interval is

$$\frac{\lambda p}{(1-p)}$$

Example 6.3.5: The Auto Dealership

- The inventory demand model for `sis4` corresponds to λ customers per week on average
- Each customer will buy
  - 0 autos with probability $1 - p$
  - 1 auto with probability $(1 - p)p$
  - 2 autos with probability $(1 - p)p^2$, etc.
- With $\lambda = 120.0$ and $p = 0.2$, average demand is 30.0

$$30.0 = \frac{\lambda p}{1-p} = \lambda \sum_{x=0}^{\infty} x(1-p)p^x = \underbrace{\lambda(1-p)p}_{19.2000} + \underbrace{2\lambda(1-p)p^2}_{7.680} + \underbrace{3\lambda(1-p)p^3}_{2.304} + \cdots$$

  - $\lambda(1 - p) = 96.0$ customers buy 0 autos
  - $\lambda(1 - p)p = 19.200$ customers buy 1 auto
  - $\lambda(1 - p)p^2 = 3.840$ customers buy 2 autos
  - $\lambda(1 - p)p^3 = 0.768$ customers buy 3 autos, etc.

Truncation

- In the previous example, no bound on number of autos purchased
- Can be made more realistic by *truncating* possible values
- Start with random variable X with possible values $\mathcal{X} = \{0, 1, 2, \ldots\}$ and cdf $F(x) = Pr(X \leq x)$
- Want to restrict X to the finite range $0 \leq a \leq x \leq b < \infty$
- If $a > 0$, $\quad \alpha = \Pr(X < a) = \Pr(X \leq a - 1) = F(a - 1)$
- $\beta = \Pr(X > b) = 1 - \Pr(X \leq b) = 1 - F(b)$
- $\Pr(a \leq X \leq b) = \Pr(X \leq b) - \Pr(X < a) = F(b) - F(a - 1)$

  Essentially, always true iff $F(b) \cong 1.0$ and $F(a - 1) \cong 0.0$

Specifying truncation points

- If a and b are specified
  - Left-tail, right-tail probabilities α and β obtained using cdf

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

  - Transformation is exact
- If α and β are specified
  - Idf can be used to obtain a and b

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

  - Transformation is not exact because X is discrete

$$\Pr(X < a) \leq \alpha \qquad \text{and} \qquad \Pr(X > b) < \beta$$

Example 6.3.6

For the *Poisson*(50) random variable I, determine a, b so that

$$\Pr(a \leq I \leq b) \cong 1.0$$

- Use $\alpha = \beta = 10^{-6}$
- Use `rvms` to compute

Determining a, b

```
a = idfPoisson(50.0,α);        /*α = 10^-6*/
b = idfPoisson(50.0,1.0 - β);  /*β = 10^-6*/
```

- Results: $a = 20$ and $b = 87$
- Consistent with the bounds produced by the conversion:

  $\Pr(I < 20) = \texttt{cdfPoisson}(50.0, 19) \cong 0.48 \times 10^{-6} < \alpha$

  $\Pr(I > 87) = 1.0 - \texttt{cdfPoisson}(50.0, 87) \cong 0.75 \times 10^{-6} < \beta$

Effects of Truncation

- Truncating *Poisson*(50) to the range $\{20, \ldots, 87\}$ is insignificant: truncated and un-truncated random variables have (essentially) the same distribution
- Truncation is useful for efficiency:
  - When idf is complex, inversion requires cdf search
  - cdf values are typically stored in an array
  - Small range gives improved space/time efficiency
- Truncation is useful for realism:
  - Prevents arbitrarily large values possible from some variates
- In some applications, truncation is significant
  - Produces a new random variable
  - Must be done correctly

  0-13-142917-5

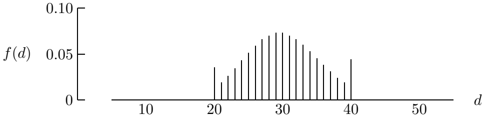
Incorrect Truncation

- Use a *Poisson*(30) demand model in program `sis2`
- Truncate the demand to the range $20 \le d \le 40$

Incorrect Truncation

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

- Original left and right tails grouped together at 20 and 40



- This is *incorrect* for most applications

Truncation by cdf Modification (1)

Example 6.3.8: Truncate *Poisson*(30) demands to range $20 \le d \le 40$

- The *Poisson*(30) pdf is (before truncation)

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

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

- Compute a new truncated random variable D_t with pdf $f_t(d)$

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

Truncation by cdf Modification (2)

- The corresponding truncated cdf is

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

- Mean and standard deviation of D_t

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

- Mean and standard deviation of *Poisson*(30)

$$\mu = 30.0 \qquad \text{and} \qquad \sigma = \sqrt{30} \cong 5.477$$

  0-13-142917-5

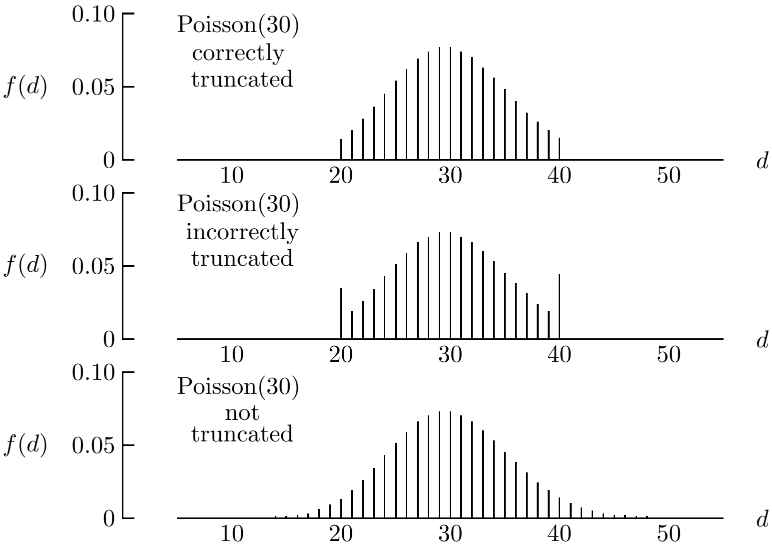
Truncation by cdf Modification (3)

- A random variate truncated to $20 \leq d \leq 40$ can be generated by inversion, using the truncated cdf $F_t(\cdot)$ and Alg.6.2.2

Truncation by cdf Modification

```
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;
```

Illustration of pdfs



Truncation By cdf Modification In General

- To truncate (integer-valued, discrete) X to possible values $\mathcal{X}_t = \{a, a+1, \cdots, b\} \subset \mathcal{X}$

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

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

- Above equations assume $a - 1 \in \mathcal{X}$
- Random values of X_t can be generated using inversion and Alg.6.2.2 with cdf $F_t(\cdot)$

Truncation by Constrained Inversion

- Use the idf of X to generate X_t truncated to $a \leq x \leq b$

Truncation by Constrained Inversion

```
/* assumes a - 1 is a possible value of X */
α = F(a-1);
β = 1.0 - F(b);
u = Uniform(α, 1.0 - β);
x = F*(u);    /* F*(·) is the idf of X */
return x;
```

- The key is that u is *constrained* to a subrange $(\alpha, 1 - \beta) \subset (0, 1)$
- Truncation is automatically enforced prior to inversion

0-13-142917-5

Example 6.3.9

- Generate a *Poisson*(30) random demand truncated to $20 \leq d \leq 40$

Example 6.3.9

```
α = cdfPoisson(30.0, 19); /*set-up*/
β = 1.0 - cdfPoisson(30.0, 40); /*set-up*/
u = Uniform(α, 1.0 - β);
d = idfPoisson(30.0, u);
return d;
```

- Uses library `rvms`
- α and β are `static` variables that are computed once only

Truncation By Acceptance-Rejection

- Truncate *Poisson*(30) by using *acceptance-rejection*

Truncation By Acceptance-Rejection

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

- Acceptance-rejection is not synchronized or monotone even if the un-truncated generator has these properties
- Truncation by cdf modification or constrained inversion is preferable

