Section 7.2: Generating Continuous Random Variates

Discrete-Event Simulation: A First Course

©2006 Pearson Ed., Inc. 0-13-142917-5
The *inverse distribution function (idf)* of $X$ is the function $F^{-1} : (0, 1) \rightarrow \mathcal{X}$ for all $u \in (0, 1)$ as

$$F^{-1}(u) = x$$

where $x \in \mathcal{X}$ is the unique possible value for $F(x) = u$

There is a one-to-one correspondence between possible values $x \in \mathcal{X}$ and cdf values $u = F(x) \in (0, 1)$

- Assumes the cdf is strictly monotone increasing
- True if $f(x) > 0$ for all $x \in \mathcal{X}$
Unlike the a discrete random variable, the idf for a continuous random variable is a true inverse

Can sometimes determine the idf in “closed form” by solving $F(x) = u$ for $x$
Examples

- If $X$ is $\text{Uniform}(a, b)$, $F(x) = (x - a)/(b - a)$ for $a < x < b$
  
  $$x = F^{-1}(u) = a + (b - a)u \quad 0 < u < 1$$

- If $X$ is $\text{Exponential}(\mu)$, $F(x) = 1 - \exp(-x/\mu)$ for $x > 0$
  
  $$x = F^{-1}(u) = -\mu \ln(1 - u) \quad 0 < u < 1$$

- If $X$ is a continuous variable with possible value $0 < x < b$ and pdf $f(x) = 2x/b^2$, the cdf is $F(x) = (x/b)^2$
  
  $$x = F^{-1}(u) = b\sqrt{u} \quad 0 < u < 1$$
Random Variate Generation By Inversion

- $X$ is a continuous random variable with idf $F^{-1}(\cdot)$
- Continuous random variable $U$ is $\text{Uniform}(0, 1)$
- $Z$ is the continuous random variable defined by $Z = F^{-1}(U)$

**Theorem (7.2.1)**

$Z$ and $X$ are identically distributed

**Algorithm 7.2.1**

If $X$ is a continuous random variable with idf $F^{-1}(\cdot)$, a continuous random variate $x$ can be generated as

```plaintext
u = Random();
return $F^{-1}(u)$;
```
Example 7.2.4: Generating a Uniform$(a, b)$ Random Variate

```c
u = Random();
return a + (b - a) * u;
```

Example 7.2.5: Generating an Exponential$(\mu)$ Random Variate

```c
u = Random();
return -\mu * \log(1 - u);
```

- $U$ is Uniform$(0, 1)$ iff $1 - U$ is Uniform$(0, 1)$
- Can generate Exponential$(\mu)$ with
  ```c
  u = Random();
  return -\mu * \log(u);
  ```
- Because this algorithm has reverse monotonicity, the algorithm in Example 7.2.5 is preferred.
Algorithms in Example 7.2.4 and 7.2.5 are ideal
Both are portable, exact, robust, efficient, clear, synchronized, and monotone
It is not always possible to solve for a continuous random variable idf explicitly by algebraic techniques
Two other options may be available
  - Use a function that accurately approximates $F^{-1}(\cdot)$
  - Determine the idf by solving $u = F(x)$ numerically
If \( Z \) is a \emph{Normal}(0, 1), the cdf is the special function \( \Phi(\cdot) \).

The idf \( \Phi^{-1}(\cdot) \) cannot be evaluated in closed form.

The idf can be \emph{approximated} as the ratio of two fourth degree polynomials (Odeh and Evans, 1974).

The approximation is efficient and essentially has negligible error.
Approximation of $\Phi(\cdot)$

- For any $u \in (0, 1)$, a $\text{Normal}(0, 1)$ idf approximation is
  $\Phi^{-1}(u) \simeq \Phi_a^{-1}(u)$ where

  $$
  \Phi_a^{-1}(u) = \begin{cases} 
  -t + p(t)/q(t) & 0.0 < u < 0.5 \\
  t - p(t)/q(t) & 0.5 \leq u < 1.0 
  \end{cases}
  $$

  and

  $$
  t = \begin{cases} 
  \sqrt{-2 \ln(u)} & 0.0 < u < 0.5 \\
  \sqrt{-2 \ln(1 - u)} & 0.5 \leq u < 1.0 
  \end{cases}
  $$

  and

  $$
  p(t) = a_0 + a_1 t + \cdots + a_4 t^4
  $$

  $$
  q(t) = b_0 + b_1 t + \cdots + b_4 t^4
  $$

  - The ten coefficients can be chosen to produce an absolute
  error less than $10^{-9}$ for all $0.0 < u < 1.0$
Example 7.2.6

Inversion can be used to generate $\textit{Normal}(0, 1)$ variates:

Example: 7.2.6: Generating a $\textit{Normal}(0, 1)$ Random Variate

```plaintext
u = Random();
return $\Phi^{-1}(u)$;
```

- This algorithm is portable, essentially exact, robust, reasonably efficient, synchronized and monotone
- Clarity?
If \( U_1, U_2, \ldots, U_{12} \) is an iid sequence of Uniform\((0, 1)\),

\[
Z = U_1 + U_2 + \ldots + U_{12} - 6
\]

is approximately \( \text{Normal}(0, 1) \)
- The mean is 0.0 and the standard deviation is 1.0
- Possible values are \(-6.0 < z < 6.0\)
- Justification is provided by the central limit theorem (Section 8.1)
- This algorithm is **not**: exact, synchronized or monotone

This algorithm is: portable, robust, relatively efficient and clear
If $U_1$ and $U_2$ are independent $Uniform(0, 1)$ RVs then

$$Z_1 = \sqrt{-2 \ln(U_1)} \cos(2\pi U_2)$$

and

$$Z_2 = \sqrt{-2 \ln(U_1)} \sin(2\pi U_2)$$

will be independent $Normal(0, 1)$ RVs (Box and Muller, 1958)

- This algorithm is: portable, exact, robust and relatively efficient;
- This algorithm is **not**: clear or monotone
- The algorithm is synchronized only in pair-wise fashion
Random variates corresponding to $Normal(\mu, \sigma)$ and $Lognormal(a, b)$ can be generated by using a $Normal(0, 1)$ random variate generator.

**Example 7.2.7: Generating a $Normal(\mu, \sigma)$ Random Variate**

```c
z = Normal(0.0, 1.0);
return \mu + \sigma \times z;
/* see Definition 7.1.7 */
```

**Example 7.2.8: Generating a $Lognormal(a, b)$ Random Variate**

```c
z = Normal(0.0, 1.0);
return exp(a + b \times z);
/* see Definition 7.1.8 */
```

Both algorithms are essentially ideal.
Numerical inversion provides another way to generate continuous random variates; that is, \( u = F(x) \) can be solved for \( x \) iteratively.

Newton’s method provides a good compromise between rate of convergence and robustness.

Given \( u \in (0, 1) \), let \( t \) be close to the value of \( x \) for which \( u = F(x) \).

If \( F(\cdot) \) is expanded in a Taylor’s series about the point \( t \)

\[
F(x) = F(t) + F'(t)(x - t) + \frac{1}{2!} F''(t)(x - t)^2 + \cdots
\]

Recall \( F'(t) = f(t) \).

For small \( |x - t| \), ignore \( (x - t)^2 \) and higher order terms.
Newton’s Method

- Set \( u = F(x) \approx F(t) + f(t)(x - t) \) and solve for \( x \) to obtain
  \[
  x \approx t + \frac{u - F(t)}{f(t)}
  \]
- Use initial guess \( t_0 \) and iterate to solve for \( x \): \( t_i \rightarrow x \) as \( i \rightarrow \infty \)
  \[
  t_{i+1} = t_i + \frac{u - F(t_i)}{f(t_i)} \quad i = 0, 1, 2, \ldots
  \]
Two Issues Relative to Newton’s Method

- The choice of an initial value $t_0$
  - The best choice for the initial value is the mode
  - For most continuous RVs described in text, $t_0 = \mu$ is an essentially equivalent choice

- The test for convergence
  - Given a convergence parameter $\epsilon > 0$
  - Iterate until $|t_{i+1} - t_i| < \epsilon$
Algorithm 7.2.2

Given $u \in (0, 1)$, the pdf $f(\cdot)$, the cdf $F(\cdot)$ and a convergence parameter $\epsilon > 0$, this algorithm will solve for $x = F^{-1}(u)$

$$x = \mu; \text{ /*} \mu \text{ is } E[X]\text{ */}$$

```plaintext
do {
    \begin{align*}
    t &= x; \\
    x &= t + (u - F(t)) / f(t);
    \end{align*}
} while (|x-t| > \epsilon);
```

return $x; \text{ /*} x \text{ is } F^{-1}(u)\text{ */}$

- If $u$ is small and $X$ is non-negative, a negative value of $x$ may occur early in the iterative process.
- Negative $t$ will cause $F(t)$ and $f(t)$ to be undefined for positive RVs.
The following modification can be used to avoid the problem

Modified Algorithm 7.2.2

\[ x = \mu; \quad /* \mu \text{ is } E[X] */ \]
do {
    \[ t = x; \]
    \[ x = t + (u - F(t)) / f(t); \]
    if (x &lt;= 0.0)
        \[ x = 0.5 * t; \]
} while (|x-t| &gt; \( \varepsilon \));
return x; /* x is \( F^{-1}(u) \)*/

Algorithms 7.2.1 and 7.2.2 together provide a general purpose inversion approach to continuous random variate generation

E.g., the \( \text{Erlang}(n, b) \) idf function in \texttt{rvms} is based on Alg.7.2.2 and can be used with Algorithm 7.2.1
Erlang Random Variates

An Erlang($n, b$) random variate can be generated by summing $n$ Exponential($b$) random variates

Generating an Erlang($n, b$) Random Variate

```c
x = 0.0;
for (i = 0; i < n; i++)
    x += Exponential(b);
return x;
```

- The algorithm is: portable, exact, robust, and clear
- The algorithm is not efficient (it is $O(n)$), synchronized or monotone
To increase computational efficiency, use

### Generating an *Erlang*(n, b) Random Variate

\[
t = 1.0;
\]
\[
\text{for } (i = 0; i < n; i++)
\]
\[
\quad t *= (1.0 - \text{Random}());
\]
\[
\text{return } -b * \log(t);
\]

- This algorithm requires only one \(\log()\) evaluation, rather than \(n\)
- Can further improve efficiency by using \(t *= \text{Random}();\)
- The algorithm remains \(\mathcal{O}(n)\), so is not efficient if \(n\) is large
If $n$ is an even positive integer, an $Erlang(n/2, 2)$ random variate is equivalent to a $Chisquare(n)$ random variable.

$X$ is a $Chisquare(n)$ random variable iff

$X = Z_1^2 + Z_2^2 + \cdots + Z_n^2$ where $Z_1, Z_2, \ldots, Z_n$ are iid $Normal(0, 1)$ random variables.

### Generating a $Chisquare(n)$ Random Variate

```c
x = 0.0;
for (i = 0; i < n; i++){
    z = Normal(0.0, 1.0);
    x += (z * z);
}
return x;
```

- The algorithm is: portable, exact, robust, clear
- The algorithm is **not**: efficient (it is $O(n)$), synchronized or monotone
Student Random Variates

$X$ is $\text{Student}(n)$ iff $X = Z / \sqrt{V / n}$ where
- $Z$ is $\text{Normal}(0, 1)$
- $V$ is $\text{Chisquare}(n)$
- $Z$ and $V$ are independent

### Generating a $\text{Student}(n)$ Random Variate

```java
z = Normal(0.0, 1.0);
v = Chisquare(n);
return z / sqrt(v / n);
```

- The algorithm is: portable, exact, robust, clear
- The algorithm is **not** synchronized or monotone
- Efficiency depends on algs. used for $\text{Normal}$ and $\text{Chisquare}$
A natural way to do this at the computational level is:

- use the algorithm to generate a sample of $n$ random variates
- and construct a $k$-bin continuous-data histogram with bin width $\delta$
- $\hat{f}$ is the histogram density and $f(x)$ is the pdf

$$\hat{f} \rightarrow f(x) \quad \text{as} \quad n \rightarrow \infty \quad \text{and} \quad \delta \rightarrow 0$$

In practice, using a large but finite value of $n$ and a small but non-zero value of $\delta$, perfect agreement between $\hat{f}(x)$ and $f(x)$ will not be achieved

- In the discrete case, it is due to natural sampling variability
- In the continuous case, the *quantization error* associated with binning the sample is an additional factor
Let $B = [m - \delta/2, m + \delta/2]$ be a small histogram bin

Use the Taylor expansion of $f(x)$ at $x = m$

$$f(x) = f(m) + f'(m)(x-m) + \frac{1}{2!} f''(m)(x-m)^2 + \frac{1}{3!} f'''(m)(x-m)^3 + \cdots$$

The probability of falling within the bin is

$$\Pr(x \in B) = \int_B f(x) \, dx = \cdots = f(m)\delta + \frac{1}{24} f''(m)\delta^3 + \cdots$$
For all $x \in B$, the histogram density is

$$\hat{f}(x) = \frac{1}{\delta} \Pr(X \in B) \approx f(m) + \frac{1}{24} f''(m)\delta^2$$

Unless $f''(m) = 0$, there is a positive or negative bias between

- $\hat{f}(x)$, the experimental density of the histogram bin and
- $f(m)$, the theoretical pdf evaluated at the bin midpoint

This bias may be significant if the curvature of the pdf is large at the bin midpoint
Example 7.2.9

- $X$ is a continuous random variable with pdf
  \[ f(x) = \frac{2}{(x + 1)^3} \quad x > 0 \]

- The cdf $X$ is
  \[ F(x) = \int_0^x f(t)dt = 1 - \frac{1}{(x + 1)^2} \quad x > 0 \]

- The idf is
  \[ F^{-1}(u) = \frac{1}{\sqrt{1 - u}} - 1 \quad 0 < u < 1 \]

- Note the pdf curvature is very large close to $x = 0$; therefore, the histogram will not match the pdf well for the bins close to $x = 0$
Random variates for $X$ can be generated using inversion.

Correctness of the inversion can be tested by constructing a histogram.

Using histogram bin widths of $\delta = 0.5$, as $n \to \infty$, $\hat{f}(x)$ and $f(m)$ are (with $d.ddd$ precision):

\[
\begin{array}{cccccccc}
m & 0.25 & 0.75 & 1.25 & 1.75 & 2.25 & 2.75 & \ldots \\
\hat{f}(x) & 1.1111 & 0.3889 & 0.1800 & 0.0978 & 0.0590 & 0.0383 \\
f(m) & 1.0240 & 0.3732 & 0.1756 & 0.0962 & 0.0583 & 0.0379 \\
\end{array}
\]

For the first bin ($m = 0.25$), the curvature bias is

\[
\frac{1}{24} f''(m) \delta^2 = 0.08192
\]
Testing for Correctness using the Empirical cdf

- Compare the empirical cdf (section 4.3) with the population cdf $F(x)$
- Eliminates binning quantization error
- For large samples (as $n \to \infty$), $\hat{F}(x) \to F(x)$
Contains 7 continuous random variate generators

- double Chisquare(long n)
- double Erlang(long n, double b)
- double Exponential(double µ)
- double Lognormal(double a, double b)
- double Normal(double µ, double σ)
- double Student(long n)
- double Uniform(double a, double b)