Discrete-Event Simulation:

A First Course

Section 6.2: Generating Discrete Random Variates

0-13-142917-5

Section 6.2: Generating Discrete Random Variates

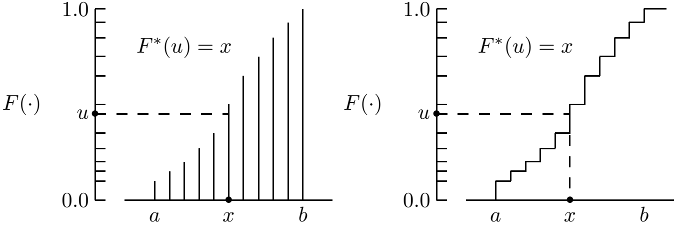
- The *inverse distribution function (idf)* of X is the function $F^* : (0,1) \rightarrow \mathcal{X}$ for all $u \in (0,1)$ as

$$F^*(u) = \min_x \{x : u < F(x)\}$$

  $F(\cdot)$ is the cdf of X
- That is, if $F^*(u) = x$, x is the smallest possible value of X for which $F(x)$ is greater than u

Example 6.2.1

- Two common ways of plotting the same cdf with $\mathcal{X} = \{a, a+1, \cdots, b\}$



Theorem 6.2.1

Theorem (6.2.1)

Let $\mathcal{X} = \{a, a+1, \cdots, b\}$ where b may be ∞ and $F(\cdot)$ be the cdf of X For any $u \in (0,1)$,

- *if $u < F(a)$, $F^*(u) = a$*
- *else $F^*(u) = x$ where $x \in \mathcal{X}$ is the unique possible value of X for which $F(x-1) \leq u < F(x)$*

Algorithm 6.2.1

- For $\mathcal{X} = \{a, a+1, \cdots, b\}$, the following linear search algorithm defines $F^*(u)$

Algorithm 6.2.1

```
x = a;
while (F(x) <= u)
    x++;
return x; /*x is F*(u)*/
```

- Average case analysis:
  - Let Y be the number of `while` loop passes
  - $Y = X - a$
  - $E[Y] = E[X - a] = E[X] - a = \mu - a$

Algorithm 6.2.2

- Idea: start at a more likely point
- For $\mathcal{X} = \{a, a+1, \cdots, b\}$, a more efficient linear search algorithm defines $F^*(u)$

Algorithm 6.2.2

```
x = mode; /*initialize with the mode of X */
if (F(x) <= u)
    while (F(x) <= u)
        x++;
else if (F(a) <= u)
    while (F(x-1) > u)
        x---;
else
    x = a;
return x; /* x is F*(u)*/
```

- For large $\mathcal{X}$, consider binary search

Idf Examples

- In some cases $F^*(u)$ can be determined explicitly
- If X is *Bernoulli*(p) and $F(x) = u$, then $x = 0$ iff $0 < u < 1 - p$:

$$F^*(u) = \begin{cases} 0 & 0 < u < 1 - p \\ 1 & 1 - p \leq u < 1 \end{cases}$$

Example 6.2.3: Idf for Equilikely

If X is *Equilikely*(a, b),

$$F(x) = \frac{x - a + 1}{b - a + 1} \quad x = a, a+1, \cdots, b$$

- For $0 < u < F(a)$, $\quad F^*(u) = a$
- For $F(a) \leq u < 1$,

$$\begin{aligned} F(x-1) \leq u < F(x) \quad &\Longleftrightarrow \quad \frac{(x-1) - a + 1}{b - a + 1} \leq u < \frac{x - a + 1}{b - a + 1} \\ &\Longleftrightarrow \quad x \leq a + (b - a + 1)u < x + 1 \end{aligned}$$

- Therefore, for all $u \in (0, 1)$

$$F^*(u) = a + \lfloor (b - a + 1)u \rfloor$$

Example 6.2.4: Idf for Geometric

If X is *Geometric*(p),

$$F(x) = 1 - p^{x+1} \quad x = 0, 1, 2, \cdots$$

- For $0 < u < F(0)$, $\qquad F^*(u) = 0$
- For $F(0) \leq u < 1$,

$$\begin{aligned} F(x-1) \leq u < F(x) \quad &\iff \quad 1 - p^x \leq u < 1 - p^{x+1} \\ &\;\;\vdots \\ &\iff \quad x \leq \frac{\ln(1-u)}{\ln(p)} < x + 1 \end{aligned}$$

- For all $u \in (0, 1)$

$$F^*(u) = \left\lfloor \frac{\ln(1-u)}{\ln(p)} \right\rfloor$$

Random Variate Generation By Inversion

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

Theorem (6.2.2)

Z and X are identically distributed

- Theorem 6.2.2 allows any discrete random variable (with known idf) to be generated with *one* call to `Random()`

Algorithm 6.2.3

If X is a discrete random variable with idf $F^*(\cdot)$, a random variate x can be generated as

```
u = Random();
return F*(u);
```


Proof for Theorem 6.2.2

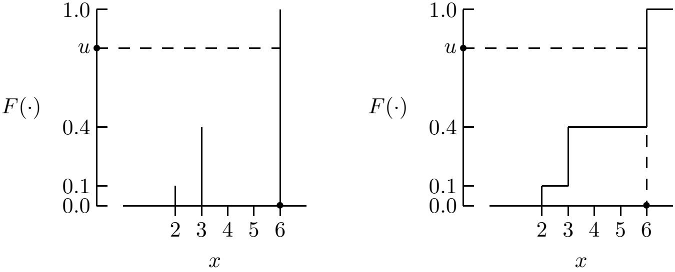
- Prove that $\mathcal{X} = \mathcal{Z}$
  - F^*: $(0,1) \to \mathcal{X}$, so $\exists\ u \in (0,1)$ such that $F^*(u) = x$
  - $Z = F^*(U)$
    It follows that $x \in \mathcal{Z}$ so $\mathcal{X} \subseteq \mathcal{Z}$
  - From definition of Z, if $z \in \mathcal{Z}$ then $\exists\ u \in (0,1)$ such that $F^*(u) = z$
  - F^*: $(0,1) \to \mathcal{X}$
    It follows that $z \in \mathcal{X}$ so $\mathcal{Z} \subseteq \mathcal{X}$
- Prove that Z and X have the same pdf
  Let $\mathcal{X} = \mathcal{Z} = \{a, a+1, \cdots, b\}$, from definition of Z and $F^*(\cdot)$ and theorem 6.2.1:
  - if $z = a$,
  $$\Pr(Z = a) = \Pr(U < F(a)) = F(a) = f(a)$$
  - if $z \in \mathcal{Z}, z \neq a$,
  $$\Pr(Z = z) = \Pr(F(z-1) \leq U < F(z)) = F(z) - F(z-1) = f(z)$$

Inversion Examples

- **Example 6.2.5** Consider X with pdf

$$f(x) = \begin{cases} 0.1 & \text{x=2} \\ 0.3 & \text{x=3} \\ 0.6 & \text{x=6} \end{cases}$$

The cdf for X is plotted using two formats



Algorithm for Example 6.2.5

Example 6.2.5

```
if (u < 0.1)
    return 2;
else if (u < 0.4)
    return 3;
else
    return 6;
```

returns 2 with probability 0.1, 3 with probability 0.3 and 6 with probability 0.6 which corresponds to the pdf of X

- This example can be made more efficient: check the ranges for u associated with $x = 6$ first (the mode), then $x = 3$, then $x = 2$
- Problems may arise when $|\mathcal{X}|$ is large or infinite

More Inversion Examples

Example 6.2.6: Generating a *Bernoulli*(p) Random Variate

```
u = Random();
if (u < 1-p)
    return 0;
else
    return 1;
```

Example 6.2.7: Generating an *Equilikely*(a, b) Random Variate

```
u = Random();
return a + (long) (u * (b - a + 1));
```

Example 6.2.8: Generating a *Geometric*(p) Random Variate

```
u = Random()
return a + (long) (log(1.0 - u) / log(p));
```

Example 6.2.9

- X is a *Binomial*(n, p) random variate

$$F(x) = \sum_{t=0}^{x} \binom{n}{x} p^x (1-p)^{n-x} \qquad x = 0, 1, 2, \cdots, n$$

- *Incomplete beta function*

$$F(x) = \begin{cases} 1 - I(x+1, n-x, p) & x = 0, 1, \cdots, n-1 \\ 1 & x = n \end{cases}$$

  Except for special cases, an incomplete beta function cannot be inverted to form a "closed form" expression for the idf

- Inversion is not easily applied to generation of *Binomial*(n, p) random variates

Algorithm Design Criteria

- The design of a correct, exact and efficient algorithm to generate corresponding random *variates* is often complex
  - Portability - implementable in high-level languages
  - Exactness - histogram of variates should converge to pdf
  - Robustness - performance should be insensitive to small changes in parameters and should work properly for all reasonable parameter values
  - Efficiency - it should be time efficient (*set-up* time and *marginal* execution time) and memory efficient
  - Clarity - it is easy to understand and implement
  - Synchronization - exactly one call to `Random` is required
  - Monotonicity - it is synchronized and the transformation from u to x is monotone increasing (or decreasing)
- Inversion satisfies some criteria, but not necessarily *all*

Example 6.2.10

- To generate *Binomial*$(10, 0.4)$, the pdf is (to $0.ddd$ precision)

x:	0	1	2	3	4	5	6	7	8
$f(x)$:	0.006	0.040	0.121	0.215	0.251	0.201	0.111	0.042	0.011

- Random variates can be generated by filling a 1000-element integer-valued array $a[\cdot]$ with 6 0's, 40 1's, 121 2's, etc.

Binomial$(10, 0.4)$ Random Variate

```
j = Equilikely(0,999);
return a[j];
```

- This algorithm is portable, robust, clear, synchronized and monotone, with small marginal execution time
- The algorithm is not exact: $f(10) = 1/9765625$
- Set-up time and memory efficiency could be problematic: for $0.ddddd$ precision, need 100 000-element array

Example 6.2.11: Exact Algorithm for *Binomial*$(10, 0.4)$

- An *exact* algorithm is based on
  - filling an 11-element floating-point array with cdf values
  - then using Alg. 6.2.2 with $x = 4$ to initialize the search
- In general, to generate *Binomial*(n, p) by inversion
  - compute a floating-point array of $n + 1$ cdf values
  - use Alg. 6.2.2 with $x = \lfloor np \rfloor$ to initialize the search
- The library `rvms` can be used to compute the cdf array by calling `cdfBinomial(n,p,x)` for $x = 0, 1, \cdots, n$
- Only drawback is some inefficiency (setup time and memory)

Example 6.2.12

- The cdf array from Example 6.2.11 can be eliminated
  - cdf values computed as needed by Alg. 6.2.2
  - Reduces set-up time and memory
  - Increases marginal execution time
- Function `idfBinomial(n,p,u)` in library `rvms` does this
- *Binomial*(n, p) random variates can be generated by inversion

Generating a Binomial Random Variate

```
u = Random();
return idfBinomial(n, p, u); /* in library rvms*/
```

- Inversion can be used for the six models:
  - Inversion is ideal for *Equilikely*(a, b), *Bernoulli*(p) and *Geometric*(p)
  - For *Binomial*(n, p), *Pascal*(n, p) and *Poisson*(μ), time and memory efficiency can be a problem if inversion is used

Alternative Random Variate Generation Algorithms

- **Example 6.2.13** Binomial Random Variates

A *Binomial*(n, p) random variate can be generated by summing an *iid Bernoulli*(p) sequence

Generating a Binomial Random Variate

```
x = 0;
for (i = 0; i < n; i++)
    x += Bernoulli(p);
return x;
```

- The algorithm is: portable, exact, robust, clear
- The algorithm is **not**: synchronized or monotone
- Marginal execution: $\mathcal{O}(n)$ complexity

Poisson Random Variates

- A *Poisson*(μ) random variable is the $n \to \infty$ limiting case of a *Binomial*($n, \mu/n$) random variable
- For large n, *Poisson*(μ) $\approx$ *Binomial*($n, \mu/n$)
- The previous $\mathcal{O}(n)$ algorithm for *Binomial*(n, p) should not be used when n is large
- The *Poisson*(μ) cdf $F(\cdot)$ is equal to an *incomplete gamma function*

$$F(x) = 1 - P(x+1, \mu) \quad x = 0, 1, 2, \cdots$$

- An incomplete gamma function cannot be inverted to form an idf
- Inversion to generate a *Poisson*(μ) requires searching the cdf as in Examples 6.2.11 and 6.2.12

Example 6.2.14

Generating a Poisson Random Variate

```
a = 0.0;
x = 0;
while (a < μ) {
    a += Exponential(1.0);
    x++;
}
return x - 1;
```

- The algorithm does not rely on inversion or the "large *n*" version of *Binomial*(n, p)
- The algorithm is: portable, exact, robust; **not** synchronized or monotone; marginal execution time can be inefficient for large μ
- It is obscure. Clarity will be provided in Section 7.3

Pascal Random Variates

- A *Pascal*(n,p) cdf is equal to an *incomplete beta function*:

$$F(x) = 1 - I(x+1,n,p) \quad x = 0,1,2,\cdots$$

- X is *Pascal*(n,p) iff $X = X_1 + X_2 + \cdots + X_n$ where $X_1, X_2, \cdots, X_n$ is an *iid Geometric*(p) sequence
- **Example 6.2.15** Summing *Geometric*(p) random variates to generate a *Pascal*(n,p) random variate

Generating a Pascal Random Variate

```
x = 0;
for(i = 0; i < n; i++)
    x += Geometric(p);
return x;
```

- The algorithm is: portable, exact, robust, clear; **not** synchronized or monotone; marginal execution complexity is $\mathcal{O}(n)$

Library rvgs

- Includes 6 discrete random variate generators (as below) and 7 continuous random variate generators
  - `long Bernoulli(double` p`)`
  - `long Binomial(long` n`, double` p`)`
  - `long Equilikely(long` a`, long` b`)`
  - `long Geometric(double` p`)`
  - `long Pascal(long` n`, double` p`)`
  - `long Poisson(double` μ`)`
- Functions `Bernoulli`, `Equilikely`, `Geometric` use inversion; essentially ideal
- Functions `Binomial`, `Pascal`, `Poisson` do *not* use inversion

Library rvms

- Provides accurate pdf, cdf, idf functions for many random variates
- Idfs can be used to generate random variates by inversion
- Functions `idfBinomial`, `idfPascal`, `idfPoisson` may have high marginal execution times
- Not recommended when many observations are needed due to time inefficiency
- Array of cdf values with inversion may be preferred