Section 6.2: Generating Discrete Random Variates
The *inverse distribution function (idf)* of $X$ is the function $F^*: (0, 1) \to \mathcal{X}$ 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

Let $\mathcal{X} = \{a, a+1, \cdots, b\}$ where $b$ may be $\infty$ 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 $F(x-1) \leq u < F(x)$
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 $\mathcal{X} = \{a, a + 1, \ldots, b\}$, a more efficient linear search algorithm defines $F^*(u)$

```plaintext
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
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 $Equilikey(a, b)$,

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

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

$$F(x - 1) \leq u < F(x) \iff \frac{(x - 1) - a + 1}{b - a + 1} \leq u < \frac{x - a + 1}{b - a + 1}$$

$$\iff x \leq a + (b - a + 1)u < x + 1$$

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 $\text{Geometric}(p)$,

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

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

$$F(x-1) \leq u < F(x) \iff 1 - p^x \leq u < 1 - p^{x+1}$$

\[ \vdots \]

$$\iff x \leq \frac{\ln(1-u)}{\ln(p)} < x + 1$$

- 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 $\text{Uniform}(0, 1)$
- $Z$ 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

$u = \text{Random}()$;
return $F^*(u)$;
Proof for Theorem 6.2.2

- Prove that $X = Z$
  - $F^*: (0, 1) \to X$, so $\exists u \in (0, 1)$ such that $F^*(u) = x$
  - $Z = F^*(U)$
    - It follows that $x \in Z$ so $X \subseteq Z$
  - From definition of $Z$, if $z \in Z$ then $\exists u \in (0, 1)$ such that $F^*(u) = z$
  - $F^*: (0, 1) \to X$
    - It follows that $z \in X$ so $Z \subseteq X$

- Prove that $Z$ and $X$ have the same pdf
  - Let $X = Z = \{a, a+1, \ldots, b\}$, 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 $z \in Z, z \neq a$,
      \[
      \Pr(Z = z) = \Pr(F(z-1) \leq U < F(z)) = F(z) - F(z-1) = f(z)
      \]
**Example 6.2.5** Consider $X$ with pdf

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

The cdf for $X$ is plotted using two formats.
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 \textit{Bernoulli}(p) Random Variate

\begin{verbatim}
    u = Random();
    if (u < 1-p)
        return 0;
    else
        return 1;
\end{verbatim}

Example 6.2.7: Generating an \textit{Equilikely}(a, b) Random Variate

\begin{verbatim}
    u = Random();
    return a + (long) (u * (b - a + 1));
\end{verbatim}

Example 6.2.8: Generating a \textit{Geometric}(p) Random Variate

\begin{verbatim}
    u = Random();
    return a + (long) (log(1.0 - u) / log(p));
\end{verbatim}
Example 6.2.9

- \( X \) is a \( \text{Binomial}(n, p) \) random variate

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

- 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 \( \text{Binomial}(n, p) \) random variates
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 $\text{Binomial}(10, 0.4)$, the pdf is (to $0.ddd$ precision)

<table>
<thead>
<tr>
<th>$x$</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f(x)$</td>
<td>0.006</td>
<td>0.040</td>
<td>0.121</td>
<td>0.215</td>
<td>0.251</td>
<td>0.201</td>
<td>0.111</td>
<td>0.042</td>
<td>0.011</td>
</tr>
</tbody>
</table>

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

```java
j = Equilike(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 $0.dddddd$ precision, need 100 000-element array.
Example 6.2.11: Exact Algorithm for $\text{Binomial}(10, 0.4)$

- 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 $\text{Binomial}(n, p)$ 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 $\text{cdfBinomial}(n, p, x)$ for $x = 0, 1, \ldots, n$
- 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

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

- Inversion can be used for the six models:
  - Inversion is ideal for Equilike\(ly(a, b)\), Bernoulli\((p)\) and Geometric\((p)\)
  - For Binomial\((n, p)\), Pascal\((n, p)\) and Poisson\((\mu)\), time and memory efficiency can be a problem if inversion is used
Example 6.2.13 Binomial Random Variates

A \textit{Binomial}(n, p) random variate can be generated by summing an \textit{iid Bernoulli}(p) sequence

Generating a Binomial Random Variate

\begin{verbatim}
x = 0;
for (i = 0; i < n; i++)
    x += Bernoulli(p);
return x;
\end{verbatim}

- The algorithm is: portable, exact, robust, clear
- The algorithm is \textbf{not}: synchronized or monotone
- Marginal execution: $O(n)$ complexity
A Poisson($\mu$) random variable is the $n \to \infty$ limiting case of a Binomial($n, \mu/n$) random variable.

For large $n$, Poisson($\mu$) $\approx$ Binomial($n, \mu/n$)

The previous $O(n)$ algorithm for Binomial($n, p$) should not be used when $n$ is large.

The Poisson($\mu$) cdf $F(\cdot)$ is equal to an incomplete gamma function:

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

An incomplete gamma function cannot be inverted to form an idf.

Inversion to generate a Poisson($\mu$) requires searching the cdf as in Examples 6.2.11 and 6.2.12.
Example 6.2.14

Generating a Poisson Random Variate

\[ a = 0.0; \]
\[ x = 0; \]
\[ \text{while } (a < \mu) \{ \]
\[ \quad a += \text{Exponential}(1.0); \]
\[ \quad x++; \]
\[ \} \]
\[ \text{return } x - 1; \]

- The algorithm does not rely on inversion or the “large } n\text{” version of } Binomial(n, p)\n- The algorithm is: portable, exact, robust; not synchronized or monotone; marginal execution time can be inefficient for large } \mu\n- It is obscure. Clarity will be provided in Section 7.3
A Pascal\((n, p)\) cdf is equal to an incomplete beta function:

\[
F(x) = 1 - I(x + 1, n, p) \quad x = 0, 1, 2, \ldots
\]

\(X\) is Pascal\((n, p)\) iff \(X = X_1 + X_2 + \cdots + X_n\) where \(X_1, X_2, \ldots, X_n\) 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

\[
x = 0;
\]

\[
\text{for}(i = 0; i < n; i++)
\]

\[
\quad x += \text{Geometric}(p);
\]

\[
\text{return } x;
\]

The algorithm is: portable, exact, robust, clear; not synchronized or monotone; marginal execution complexity is \(O(n)\)
Library rvg

- 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 Equil likely(long \( a \), long \( b \))
  - long Geometric(double \( p \))
  - long Pascal(long \( n \), double \( p \))
  - long Poisson(double \( \mu \))

- Functions Bernoulli, Equillikely, 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