IEOR E4404 Handout #5
G. Iyengar February 6th, 2002

Lecture 3
Inverse transform method

- How does one transform a sample of the uniform[0,1] random variable into a sample of a given distribution ?

- Methods for transformation
  - Inverse transform method
  - Convolution method
  - Composition method
  - Acceptance-rejection method

Generating random numbers

Problem: Generate sample of a random variable X with a given density f. (The sample is called a *random variate*)

What does this mean ?

Answer: Develop an algorithm such that if one used it repeatedly (and independently) to generate a sequence of samples $X_1, X_2, \ldots, X_n$ then as n becomes large, the proportion of samples that fall in any interval $[a,b]$ is close to $\mathbf{P}(X \in [a,b])$, i.e.

$$\frac{\#\{X_i \in [a,b]\}}{n} \approx \mathbf{P}(X \in [a,b])$$

Solution: 2-step process

- Generate a random variate uniformly distributed in $[0,1]$ .. also called a *random number*
- Use an appropriate transformation to convert the random number to a random variate of the correct distribution

why is this approach good ?

Answer: Focus on generating samples from $\mathbf{ONE}$ distribution only.

Pseudo-random numbers

Many different methods of generating a (uniform[0,1]) random number ...

1. physical methods : roulette wheel, balls for the lottery, electrical circuits etc.
2. numerical/arithmetic : sequential method ... each new number is a deterministic function of the past numbers

For simulation ... numerical method works best

- numbers generated by the numerical method are never *random* !
- enough that the numbers "look" uniformly distributed and have no correlation between them i.e. pass *statistical tests*
- All we want is that the central limit theorem should kick in ...

Properties that psuedo-random number generators should possess

1. it should be fast and not memory intensive
2. be able to reproduce a given stream of random numbers ... why ?
   - debugging or verification of computer programs
   - may want to use *identical* numbers to compare different systems
3. provision for producing several different independent "streams" of random numbers

Linear Congruential generators

- Will not focus too much on generating random numbers ... all simulators and languages have good generators

- A good example ... **Linear congruential generators**

$$Z_i = (aZ_{i-1} + c)\,(\text{mod } m) \quad i \geq 1, \qquad U_i = \frac{Z_i}{m}.$$

— modulus m, multiplier a, increment c, and seed Z_0 are nonnegative

— $a < m$, and $c, Z_0 < m$

Example

$Z_i = (5Z_{i-1} + 3)(\text{mod}16)$

i	Z_i	i	Z_i	i	Z_i	i	Z_i
0	7	5	10	10	9	15	4
1	6	6	15	11	0	16	7
2	1	7	12	12	3	17	6
3	8	8	15	13	2	18	1
4	11	9	14	14	13	19	8

$Z_i = (5Z_{i-1} + 6)(\text{mod}16)$

i	Z_i	i	Z_i	i	Z_i	i	Z_i
0	7	5	1	10	3	15	13
1	9	6	11	11	5	16	7
2	3	7	13	12	15	17	9
3	5	8	7	13	1	18	3
4	15	9	9	14	11	19	5

- Several other types of generators

  — more general congruences $Z_i = g(Z_{i-1}, Z_{i-2}, \ldots)(\text{mod } m)$.

  — a popular example ... Tautsworth generator

$$Z_i = (c_1 Z_{i-1} + c_2 Z_{i-2} + \ldots + c_q Z_{i-q})(\text{mod } 2)$$

Assume that we have an algorithm (program) that generates independent samples of uniform[0,1] random variable.

Generating random variates

- Inverse transform method
- Composition approach
- Convolution method
- Acceptance-Rejection technique

Continuous random variables

- Generate a continuous random variable $X \sim F$ as follows :

> 1. Generate a uniform random variable U
> 2. Set $X = F^{-1}(U)$

(Assumption: The inverse $F^{-1}(x)$ exists ...)

Using the inverse and hence ... **Inverse Transform Method**

- Proof: Have to show that the CDF of the samples produced by this method is precisely $F(x)$.

$$\begin{aligned} \mathbf{P}(X \leq x) &= \mathbf{P}(F^{-1}(U) \leq x) \\ &= \mathbf{P}(U \leq F(x)) \qquad (1) \\ &= F(x) \qquad (2) \end{aligned}$$

where

  - (1) follows by the fact that F is an increasing function
  - (2) follows from the fact $0 \leq F(x) \leq 1$ and the CDF of a uniform $F_U(y) = y$ for all $y \in [0, 1]$

- Algorithm:
  - Given: the CDF $F(x)$ or the density $f(x)$. If density given then first integrate to get CDF. (Most frequent error: incorrect limits on the integration)
  - Set $F(X) = U$ and solve for X in terms of U. Have to make sure that the solution X lies in the correct range.

Example : exp(λ)

- The density $f(x) = \lambda e^{-\lambda x}$ and the cdf $F(x) = 1 - e^{-\lambda x}$.

- Set $F(X) = U$ and solve for U

$$
\begin{aligned}
1 - e^{-\lambda X} &= U \\
e^{-\lambda X} &= 1 - U \\
X &= -\frac{1}{\lambda}\log(1-U)
\end{aligned}
$$

- Algorithm :

> 1. Generate random number U
> 2. Set $X = -\frac{1}{\lambda}\log(1-U)$

- But if U is uniform[0,1] then $1 - U$ is also uniform[0,1] so one might as well define

$$X = -\frac{1}{\lambda}\log(U)$$

Inverse transform method : Discrete random variables

- Want to generate a *discrete* random variable X with pmf

$$\mathbf{P}(X = x_i) = p_i, \quad i = 1, \ldots, m$$

- Consider the following algorithm

> 1. Generate a *random number* U
> 2. *Transform* U into X as follows,
>
> $$X = x_j \quad \text{if} \quad \sum_{i=1}^{j-1} p_i \leq U < \sum_{i=1}^{j} p_i$$

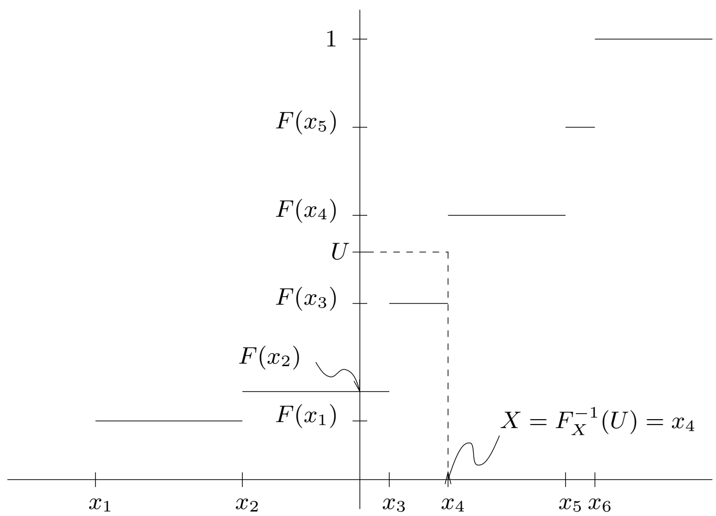
- Proof the algorithm works ...

$$\begin{aligned}\mathbf{P}(X = x_j) &= \mathbf{P}\left(\sum_{i=1}^{j-1} p_i \leq U < \sum_{i=1}^{j} p_i\right) \\ &= \sum_{i=1}^{j} p_i - \sum_{i=1}^{j-1} p_i = p_j\end{aligned}$$

QED

- Suppose $x_1 < x_2 < \ldots < x_m$ then the cdf of X F_X is

$$F_X(x) = \begin{cases} 0, & x < x_1, \\ \Sigma_{i=1}^{j} p_i, & x_j \le x < x_{j+1}, j \le m-1 \\ 1, & x \ge x_m \end{cases}$$



Thus ... $X = x_j$ if $F_X(x_{j-1}) \le U < F_X(x_j)$

- If one defines the *generalized* inverse F_X^{-1} by

$$F_X^{-1}(x) = \min\{y \mid F_X(y) \ge x\}$$

then ... $X = F_X^{-1}(U)$

Examples : simple distribution

- Want to simulate the random variable X with the pmf

$$p_1 = 0.2, p_2 = 0.15, p_3 = 0.25, p_4 = 0.40$$

- Algorithm :

1. Generate a *random number* U
2. If $U < p_1 = 0.2$, set $X \leftarrow x_1$ and stop
3. If $U < p_1 + p_2 = 0.35$, set $X \leftarrow x_2$ and stop
4. If $U < p_1 + p_2 + p_3 = 0.60$, set $X \leftarrow x_3$ and stop
5. Else set $X \leftarrow x_4$ and stop

- Is this the most efficient way of generating the random variate ?

  The amount of time is proportional to the *number of intervals* one must search. Thus, it helps to consider the values x_j in *decreasing* order of p_i's.

- More efficient algorithm :

1. Generate a *random number* U
2. If $U < p_4 = 0.4$, set $X \leftarrow x_4$ and stop
3. If $U < p_4 + p_3 = 0.65$, set $X \leftarrow x_3$ and stop
4. If $U < p_4 + p_3 + p_1 = 0.85$, set $X \leftarrow x_1$ and stop
5. Else set $X \leftarrow x_2$ and stop

Example : Geometric distribution

- In this case get a closed form expression for $F_X^{-1}(U)$.

- The pmf of the geometric random variable X is :

$$\mathbf{P}(X = i) = pq^{i-1}, \quad i \geq 1, q = (1-p)$$

- The cdf of the random variable

$$\begin{aligned} F(i) &= \mathbf{P}(X \leq i), \\ &= 1 - \mathbf{P}(X > i), \\ &= 1 - \mathbf{P}(\text{first } i \text{ trials are failures}) = 1 - q^i. \end{aligned}$$

- Transformation : $X = j$ if

$$\begin{aligned} F(j-1) \leq U < F(j) &\Leftrightarrow 1 - q^{j-1} \leq U < 1 - q^j \\ &\Leftrightarrow q^j < 1 - U \leq q^{j-1} \end{aligned}$$

Therefore

$$\begin{aligned} X &= \min\{j \mid q^j < 1 - U\} \\ &= \min\{j \mid j\log(q) < \log(1-U)\} \\ &= \min\left\{j \,\middle|\, j > \frac{\log(1-U)}{\log(q)}\right\} \end{aligned}$$

i.e.

$$X = \lceil \frac{\log(1-U)}{\log(1-p)} \rceil$$

where $\lceil z \rceil$ is the smallest integer larger than z.

Example : Binomial random variable

- No closed form solution ... algorithm
- the pmf is

$$p_i = \mathbf{P}(X = i) = \binom{n}{i} p^i (1-p)^i, \quad i = 0, \ldots, n$$

Therefore ...

$$p_i = \left(\frac{n-i}{i+1}\right)\left(\frac{p}{1-p}\right) p_{i-1}$$

- Algorithm

1. Generate a *random number* U
2. Set $i \leftarrow 0$, $P \leftarrow (1-p)^n$, $F \leftarrow P$,
3. If $U < F$, set $X \leftarrow i$ and stop
4. Set $P \leftarrow \left(\frac{n-i}{i+1}\right)\left(\frac{p}{1-p}\right) P$, $F \leftarrow F + P$, $i \leftarrow i + 1$
5. Goto step 3

Disadvantages and advantages

Disadvantages:

1. requires a closed form expression for $F(x)$
2. speed ... often very slow because a number of comparisons required

Advantages:

Inverse transform method preserves *monotonicity* and *correlation* which helps in

1. Variance reduction methods ...
2. Generating truncated distributions ...
3. Order statistics ...

Convolution method

- Suppose X is a sum of independent random variables $Z_1, Z_2, \ldots, Z_m$, i.e.

$$X = Z_1 + Z_2 + \ldots + Z_m$$

where $Z_i \sim F_i$ and are all independent.

- Algorithm :

1. Generate m random numbers U_1, U_2, ..., U_m
2. Inverse transform method : $Z_i = F_i^{-1}(U_i)$
3. Set $X = \sum_{i=1}^{m} Z_i$

- Why is this called the *convolution method* ?

Let Z_1, Z_2 independent with densities f_1 and f_2 resp.

Let $X = Z_1 + Z_2$ then the density f_X of X is given by

$$f_X(x) = \int f_1(u) f_2(x-u) du$$

Operation on f_1 and f_2 called *convolution*. Denoted by $f_1 * f_2$.
Method *samples* from the density $f_1 * f_2 * \ldots * f_m$.

Example : Erlang(λ,m)

- **Problem**: Generate a sample from Erlang(λ, m) distribution.

  The density of $f_{\lambda,m}$ of Erlang(λ, m) dist. is

$$f_{\lambda,m}(x) = \frac{\lambda(\lambda x)^{m-1}}{(m-1)!} e^{-\lambda x}$$

- **Fact :** If $Z_i \sim$ exp(λ) and independent, then

$$X = Z_1 + Z_2 + \ldots + Z_m \text{ is Erlang}(\lambda, m)$$

- All set for the convolution method ...

> 1. Generate m random numbers U_1, U_2, ..., U_m
> 2. Set $Z_i = -\frac{1}{\lambda}\log(U_i)$
> 3. Set $X = \Sigma_{i=1}^{m} Z_i$

- Not very efficient ... need m random numbers to generate 1 sample

- Will discuss a more efficient method later.