UNIT 2 PSEUDO RANDOM NUMBER GENERATION

2.0 INTRODUCTION

A pseudo-random number generation is the methodology to develop algorithms and programs that can be used in, probability and statistics applications when large quantities of random digits are needed. Most of these programs produce endless strings of single-digit numbers, usually in base 10, known as the decimal system. When large samples of pseudo-random numbers are taken, each of the 10 digits in the set {0,1,2,3,4,5,6,7,8,9} occurs with equal frequency, even though they are not evenly distributed in the sequence.

Many algorithms have been developed in an attempt to produce truly random sequences of numbers, endless strings of digits in which it is theoretically impossible to predict the next digit in the sequence based on the digits up to a given point. But the very existence of the algorithm, no matter how sophisticated, means that the next digit can be predicted! This has given rise to the term pseudo-random for such machine-generated strings of digits. They are equivalent to random-number sequences for most applications, but they are not truly random according to the rigorous definition.

A simulation that has any random aspects at all, must involve sampling or generating random variables from different probability distributions. These distributions are often specified, that is the form of the distribution functions is explicitly known, for example it can be exponential, gamma, normal or Poisson as discussed in Unit 1.

Random number generation has intrigued scientists for several years and a lot of efforts has been spent on the creation of randomness on a deterministic (non-random) machine, that is to design computer algorithms that are able to produce 'random' sequences of integers. This is not a trivial task. Such algorithms are called generators and all generators have flaws because all of them construct the n-th number in the sequence as a function of the $(n-1)$ – th number, initialized with a non-random seed value. Numerous techniques have been invented over the years that measure just how random a sequence is, and most well known generator have subjected to rigorous testing. The mathematical tools that are required to design such an algorithm are largely number theoretic and combinatorial in nature. These tools differ drastically from those needed when we want to generate sequences of integers with certain non-uniform distributions given that a perfect uniform random number generator is available.

The methodology of generating random numbers has a long and interesting history. The earliest methods were essentially carried out by hand, such as casting lots, throwing dice, dealing out cards or drawing numbered balls from a well-stirred urn. Many lotteries are still operating in this way. In the early twentieth century statisticians joined gamblers in generating random numbers and mechanized devices were built to generate random numbers more quickly. Some time later, electric circuits based on randomly pulsating vacuum tubes were developed that delivered random digits at rates up to 50 per second. One such random number generator machine the Electronic Random Number Indicator Equipment (ERNIE) was used by the British General Post Office to pick the winners in the Premium Savings Bond lottery. Another electronic device was used by the Rand Corporation to generate a table of million random digits. In India also Indian Statistical Institute has published a book just the collection of million random numbers in the mid twentieth century which was used for sample survey planning.

As computers and simulation became more widely used, increasing attention was paid to methods of random number generation compatible with the way computers work. A good uniform between 0 and 1, random generator should possess certain properties as listed below:

- Above all, the numbers produced should appear to be distributed uniformly on [0, 1] and should not exhibit any correlation with each other; otherwise, the simulation's results may be completely invalid.
- From a practical point of view a generator should be fast and avoid the need for a lot of storage.
- We should be able to produce a given stream of random numbers from a given initial (seed) value for at least two reasons. First this can sometimes make debugging or verification of the computer program easier or we might want to use identical random numbers in simulating different systems in order to obtain a more precise comparison.

In this unit we will describe how to generate $U(0,1)$ (uniform between 0 and 1, see unit 1 for the actual definition) in a

computer. Once we have a U (0, 1) random number, we will use that to generate several other deviates from different discrete and continuous distributions.

2. 1 OBJECTIVES

After reading this unit, you should know

- how to generate U (0, 1) random number in a computer
- how to generate random deviates from any discrete distribution
- how to generate random numbers from many continuous distributions, like exponential, Weibull, gamma, normal, chi-square etc.

2.2 UNIFORM RANDOM NUMBER GENERATORS

As we had mentioned in the previous section that we need the uniform random number of generator for generating random numbers from any other distributions. Therefore, it is very important to have a very good uniform random number generator. There are several methods available to generate uniform random numbers. But currently the most popular one is the linear congruential generator (LCG). Most of the existing software's today use this LCGs proposed by Lehmer in the early 50's. The LCGs provides an algorithm how to generate uniform random number between (0,1). It can be simply described as follows.

A sequence of integers $Z_1, Z_2,\ldots$ is defined by the recursive formula

$$Z_i = (aZ_{i-1} + c) \text{ (mod m)}, \qquad (1)$$

where m, the modulus, a, the multiplier, c, the increment and Z_0, the seed or the initial value, are all non-negative integers. Those who are not familiar with the definition of modules, note that for non-negative integers, x, y and z, $x = y$ (mod z) means x is the remainder when the integer y is divided the integer z. For example if $y = 10$ and $z = 3$, then $x = 1$, or if $y = 10$ and $z = 2$, then $x = 0$. Therefore, from (1) it is clear that to obtain Z_i, first divide $aZ_{i-1} + c$ by m and Z_i is the corresponding remainder of this division. It is clear that $1 \le Z_i \le m - 1$ and to obtain the desired random numbers U_i, for $i = 1, 2\ldots$. On [0, 1], we let $U_i = \frac{Zi}{m}$. The choice of the non-negative integers, a, c and m are the most crucial steps, and they come from the theoretical considerations. In addition to non-negatively, they also should satisfy $0 < m$, $a < m$ and $c < m$. Moreover, the initial value $Z_0 < m$.

Immediately, two objections could be raised against LCGs. The first objection is one which is common to all random number generators, namely the Z_i's defined by (1) are not really random. It can be easily seen that for $i = 1, 2,\ldots,$

$$Zi = \left[a^i Z_0 + \frac{c(a^i - 1)}{a - 1} \right] \text{(mod m)},$$

so that every Z_i is completely determined by m, a, c and Z_0. However, by careful choice of these four parameters the aim is to induce a behavior in the Z_i's that makes the corresponding U_i's appear to be independent identically distributed U (0, 1) random variates when subjected to variety to statistical tests.

The second objection to LCGs might be that the U_i's can take on only the rational numbers 0, $\frac{1}{m}$, $\frac{2}{m}$, …., $\frac{(m-1)}{m}$; in fact the U_i's might take only a fraction of these values, depending on the specifications of the four parameters. Therefore, there is no possibility of getting a value of U_i between, say $\frac{0.1}{m}$ and $\frac{0.9}{m}$, whereas this should occur with probability $\frac{0.8}{m} > 0$. We will see later that the modulus m is usually chosen to be very large, say 10^9 or more, so that the points in [0, 1] where U_i's can fall are very dense. This provides an accurate approximation to the true continuous U (0, 1) distribution, sufficient for most purposes.

Let us consider the following example:

Example 1 : Consider the LCG defined by $m = 16$, $a = 5$, $c = 3$ and $Z_0 = 7$. The following table gives Z_i and U_i (up to three decimal places) for $i = 1, \ldots, 19$. Note that $Z_{17} = Z_1 = 6$, $Z_{18} = Z_2 = 1$ and so on. Therefore, from $i = 17$, the sequence repeats itself. Naturally we do not seriously suggest anybody to use this generator. The main reason here m is too small. This we are presenting just for illustrative purpose.

Table 1

The LCG $Z_i = (5Z_{i-1} + 3) \text{ (mod 16)}$ with $Z_0 = 7$

i	Z_i	U_i	i	Z_i	U_i	i	Z_i	U_i	i	Z_i	U_i

0	7	-	5	10	0.625	10	9	0.563	15	4	0.250
1	6	0.375	6	5	0.313	11	0	0.000	16	7	0.438
2	1	0.063	7	12	0.750	12	3	0.188	17	6	0.375
3	8	0.500	8	15	0.938	13	2	0.125	18	1	0.063
4	11	0.688	9	14	0.875	14	13	0.813	19	8	0.500

Note that the repeating behaviour of LCG is inevitable. By the definition of Z_i, whenever it takes on a value it had taken previously, from that point onward the sequence will repeat itself endlessly. The length of a cycle is called the period of a generator. For LCG, Z_i depends only on the previous value Z_{i-1} and since $0 \le Z_i \le m - 1$, it is clear that the period is at most m. If it m, the LCG is said to have full period. Clearly, if a generator is full period, any choice of the initial seed Z_0 from {0,…,*m-1*} will produce the entire cycle in some order.

Since for large scale simulation projects may require hundreds of thousands of random numbers, it is desirable to have LCGs with long periods. Moreover, it is desirable to have full period LCGs, because then it is assured to have every integer between 0 and $m - 1$ exactly once in every cycle. Thus it is very important to know how to choose a, m and c so that the corresponding LCG will have full period. We should also keep in mind that obtaining full period is just one desirable property for a good LCG. It should also have good statistical properties, such as apparent independence, computational and storage efficiency and reproducibility. Reproducibility is simple. For reproducibility, we must only remember that the initial seed Z_0 initiates the generator with this value again to obtain the same sequence of U_i exactly. Some of the standard LCGs with different values of a, m and c are presented below. These LCG have been observed to perform very well in several machines and passed the standard statistical tests also.

Generator 1: $a = 16807$, $m = 2^{31} - 1$, $c = 0$.
Generator 2: $a = 1664525$, $m = 2^{32}$, $c = 1664525$.

Fortunately today, most of the simulation packages and even simple scientific calculators have reliable U (0 1,) generator available.

Check your progress 1

E1 What do you mean by Pseudo random number generation? What is the practical advantage of the concept of random number generation? Do you know any algorithm which works in designing the software for Random number generation?

2.3 GENERATING RANDOM VARIATES FROM ARBITRARY DISTRIBUTIONS

A simulation that has any random aspects at all must involve generating random variates from different distributions. We usually use the phrase generating a random variate to refer the activity of obtaining an observation or a realization on a random variable from the desired distribution. These distributions are often specified as a result of fitting some appropriate distributional form. They are often specified in advance, for example exponential, gamma or Poisson etc. In this section we assume that the distributional form is already been specified including the values of the parameters and we address the issue of how we can generate random variate from this specified distribution.

We will see in this section that the basic ingredient needed for every method of generating random variates from any distribution is a source of independent identically distributed U (0, 1) random variates. Therefore, it is essential that a statistically reliable U (0, 1) generator is available for generating random deviate correctly from any other distribution.Therefore, from now on we assume that we have a reliable sequence of U(0,1) variates available to us.

There are several issues which should be kept in mind before using any particular generator. The most important issue is of course the exactness. It is important that one should use an algorithm that results in random variates with exactly the desired distribution, within the unavoidable external limitations of machine accuracy and the exactness of U (0,1) random number generator. The second important issue is efficiency. Given that we have several choices, we should choose that algorithm which is efficient in terms of both storage space and execution time. Now we provide some of the most popular and standard techniques to generator non-uniform random deviates, it may be both discrete or continuous and even mixed distribution also.

2.4 INVERSE TRANSFORM

Suppose we want to generate a random variate X that has a continuous and strictly increasing distribution function F, when $0 < F(x) < 1$, i.e., whenever $x_1 < x_2$ then $F(x_1) < F(x_2)$. We draw a curve of F in the Figure 1. Let F^{-1} denote the inverse of the function F. Then an algorithm for generating a random variate X having distribution function F is as follow .

ALGORITHM

- Step 1: Generate U_1 from U_1 (0,1)
- Step 2: Return $X_1 = F^{-1}(U_1)$.

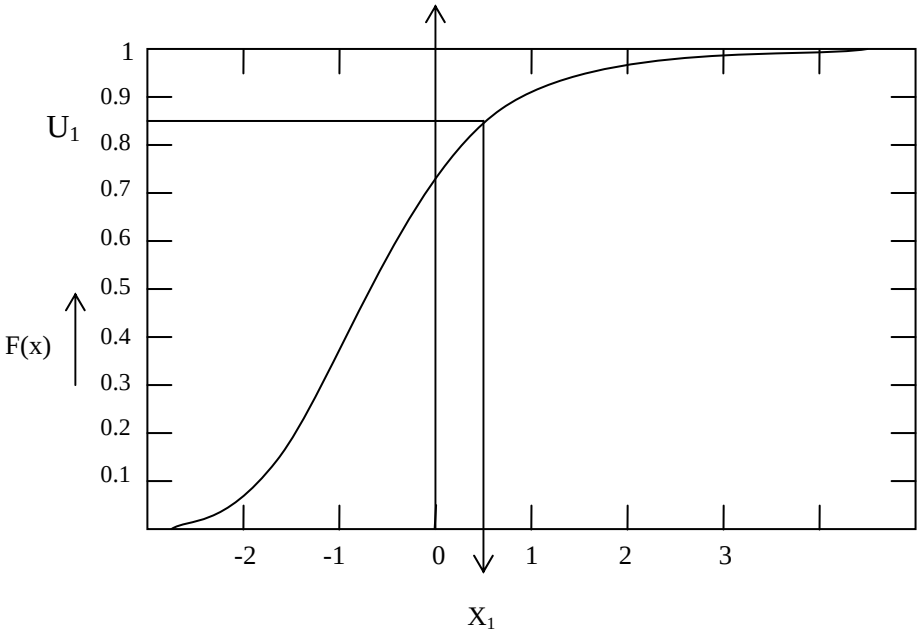


Figure 1: Distribution function of a continuous a random variable.

Note that when F is a strictly increasing function, $F^{-1}(U)$ will be always defined, since $0 \le U \le 1$ and the range of F is [0, 1]. Figure 1 illustrates the algorithm graphically. According to the figure it is clear that the uniform random variable U_1 results the random variable X_1 with the distribution function F. To show that the value X_1 returned by the above algorithm has the desired distribution function F, note that

$$P(X_1 \le x) = P(F^{-1}(U_1) \le x) = P(U_1 \le F(x)) = F(x).$$

The first equality follows from the definition of X_1, the second equality follows because F is invertible and the third equality follows because U_1 follows $U(0,1)$.

EXAMPLE 2: Let X have the Weibull distribution with the following probability density function:

$$f(x) = \begin{cases} \alpha\lambda e^{-\lambda x^{\alpha}} x^{\alpha-1} & if \quad x > 0 \\ 0 & if \quad x < 0 \end{cases} \quad \text{Find } F^{-1}$$

SOLUTION Here α and λ both are known constants and both of them are strictly greater than 0.
Therefore, X has the distribution function

$$F(x) = \begin{cases} 1 - e^{-\lambda x^{\alpha}} & if \quad x > 0 \\ 0 & if \quad x \le 0 \end{cases}$$

Therefore, to compute $F^{-1}(u)$, let us equate $u = F(x)$ and we solve for x to obtain

$$F^{-1}(u) = \left[\frac{1}{\lambda}\{-\ln(1-u)\}\right]^{\frac{1}{\alpha}}$$

Therefore to generate X from a Weibull distribution with $\alpha = 2$ and $\lambda = 1$, generate U from $U(0, 1)$ and then set

$$X = [\{-\ln(1-u)\}]^{\frac{1}{2}}$$

In this case also as before, it is possible to replace U by $1 - U$, therefore we can use

$$X = [\{-\ln u\}]^{\frac{1}{2}},$$

to avoid one subtraction.

The inverse-transform method can be used also when the random variable X is discrete.
In this case the distribution function is

$$F(x) = P(X \le x) = \sum_{x_i \le x} p(x_i),$$

Where $p(x_i)$ is the probability mass function of X, *i.e.*,

$$p(x_i) = P(X = x_i).$$

We can assume that X can take values $x_1, x_2, \ldots$, where $x_1 < x_2 < \ldots$. Then the algorithm is as follows:

ALGORITHM

- Step 1: Generate U from $U(0,1)$
- Step 2: Determine the smallest positive integer I such that $U \le F(x_1)$ and return $X = x_I$. The following figure 2 illustrates the method. In that case we generate $X = x_4$

Now to show that the discrete inverse transform method is valid, we need to show that $P(X = x_i) = p(x_i)$ for all $i = 1$, we get $X = x_1$, if and only if $U \le F(x_1) = p(x_1)$, since x_i's are in the increasing order and U follows $U(0,1)$. For $i \ge 2$, the algorithm sets $X = x_i$ if and only if $F(x_{i-1}) < U \le F(x_i)$, since the i chosen by the algorithm is the smallest positive integer such that $U \le F(x_i)$. Further, since U follows $U(0,1)$ and $0 \le F(x_{i-1}) < F(x_i) \le 1$,

$$P(X = x_i) = P\{F(x_{i-1} < U \le F(x_i)\} = F(x_i) - F(x_{i-1}) = p(x_i).$$

Now consider the following example.

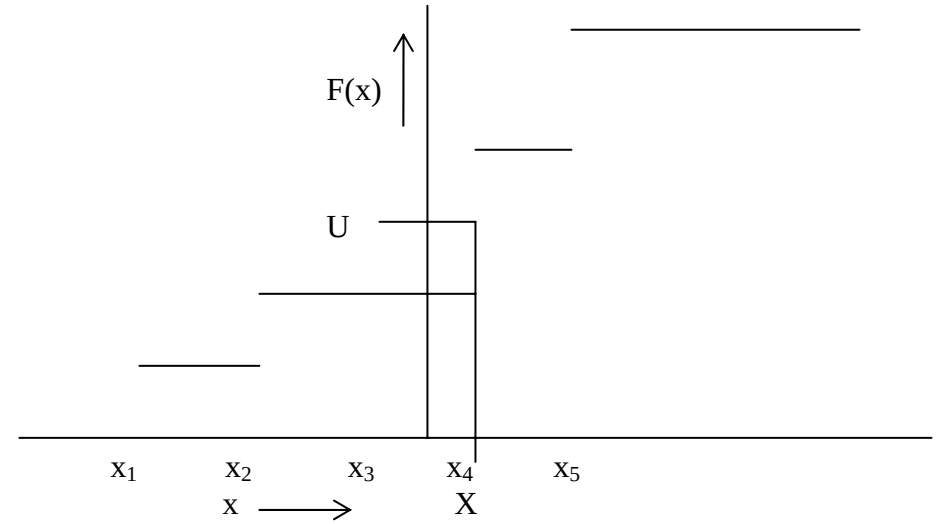


Figure 2: Distribution function of a discrete random variable

EXAMPLE 3: Suppose we want to generate a random sample from the following discrete probability distribution.
$P(X = 1) = \frac{1}{2}$, $P(X = 2) = \frac{1}{4}$, $P(X = 4) = \frac{1}{4}$. Generate a random sample from X?

SOLUTION The distribution function of the random variable X is

$$F(x) = \begin{cases} 0 & if \quad x < 1 \\ \frac{1}{2} & if \quad 1 \le x < 2 \\ \frac{3}{4} & if \quad 2 \le x < 4 \\ 1 & if \quad x \ge 4 \end{cases}$$

The distribution function is X is presented in the Figure 3. If we want to generate a random sample from X, first generate a random variable U from $U(0, 1)$. If $U \le \frac{1}{2}$, then assign $X = 1$. If $\frac{1}{2} < U \le \frac{3}{4}$, then assign $X = 2$, otherwise assign $X = 4$.

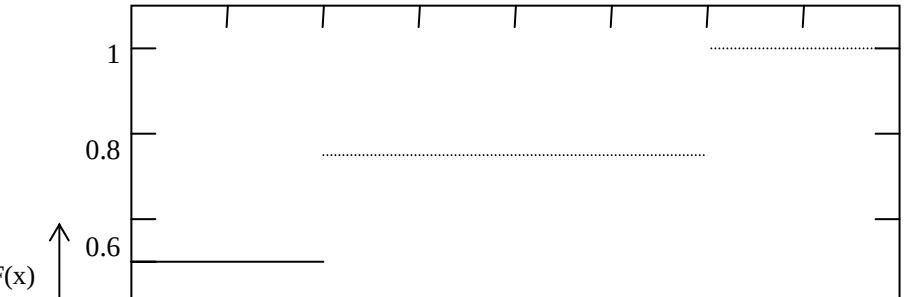


Figure 3: Distribution function of the random variable X of example 3.

Check your progress 2

E1 Let X have the exponential distribution with mean 1. The distribution function is $F(x) = \begin{cases} 1-e^{-x} & if \quad x \geq 0 \\ 0 & if \quad x < 0 \end{cases}$ Find F^{-1}

E2 Consider another discrete random variable which may take infinitely many values. Suppose the random variable X has the following probability mass function.

$P(X = i) = p_i = \frac{1}{2^i}$, i = 1, 2, 3, ……. . Generate a random sample from X?

2. 5 ACCEPTANCE-REJECTION METHOD

In the last subsection we have discussed the inverse transformation method to generate random number from different non-uniform distributions. Note that apparently, the inverse transformation method seems to be the most general method to generate random deviate from any distribution function functions. In fact it can be used provided the distribution function can be written in an explicit form, or more precisely the inverse of the distribution function can be computed analytically. For example, in case of exponential, Weibull, Cauchy distributions the distribution function and also their inverses can be constructed analytically. Unfortunately that is not the case in general. Suppose the random variable X follows gamma with the shape and scale parameters as α and λ respectively. Then the density function of X, say
$fx(x \mid \alpha, \lambda)$, is

$$f(x \mid \alpha, \lambda) = \begin{cases} \frac{\lambda^{\alpha}}{\Gamma\alpha} x^{\alpha-1} e^{-\lambda x} & if \quad x > 0 \\ 0 & if \quad x \leq 0 \end{cases}$$

The distribution function X, say $F(x \mid \alpha, \lambda) = \int_0^x f(y \mid \alpha, \lambda) dy$ can not be expressed in explicit form. Therefore, $F^{-1}(x \mid \alpha, \lambda)$ also can not be calculated explicitly. Exactly the same problem arises if X is a normal random variable. Suppose X is a normal random variable with mean μ and variance σ^2, then the probability density function of X, say $f(x \mid \mu, \sigma)$ is

$$f(x \mid \mu, \sigma) = \frac{1}{\sqrt{2\pi\sigma}} e^{-\frac{(x-\mu)2}{2\sigma^2}}, \text{ for } -\infty < x < \infty.$$

In this case also the distribution function can not be computed analytically and similarly it's inverse also. Therefore in these cases we can not apply the inverse transformation method to generate the corresponding random deviates. The acceptance-rejection method can be used quite effectively to generate these random deviates. It can be described as follows.

Suppose we have a density function $f(x)$ and we want to generate a random deviate from the density function $f(x)$. The distribution function of $f(x)$ can not be expressed in explicit form. The acceptance-rejection method requires that we specify a function $g(x)$ that majorizes the function $f(x)$, that is, $f(x) \leq g(x)$ for all x. Naturally, $g(x)$ will not be a density function always, since

$$c = \int_{-\infty}^{\infty} g(x) dx \geq \int_{-\infty}^{\infty} f(x) = 1,$$

but the function $h(x) = \frac{1}{c} g(x)$ is clearly a density function provided $c < \infty$. Now for any given $f(x)$, we choose the function $g(x)$, such that $c < \infty$ and it is possible to generate random deviate from $g(x)$ by a inverse transformation method. Then we can generate the random deviate X from $f(x)$ as follows:

ALGORITHM

- Step 1: Generate Y having density function $g(x)$.
- Step 2: Generate U from $U(0,1)$ which is independent of Y.
- Step 3: If $U \le \frac{f(Y)}{g(Y)}$, $X = Y$, otherwise go back to Step 1 and try again.

Note that the algorithm is looping back to Step 1 until we generate a pair (Y, U) pairs in Steps 1 and 2 for which $U \le \frac{f(Y)}{g(Y)}$, when we accept the value Y for X. Theoretically it can be shown that the random deviate X generated by the above algorithm has indeed the probability density function $f(x)$. Since it is not very easy to prove the result we do not provide it.

EXAMPLE 4**:** Consider the following density function

$$f(x) = \begin{cases} 60x^3(1-x)^2 & if \quad 0 \le x \le 1 \\ 0 & \text{otherwise} \end{cases}$$

In this case the distribution function of $f(x)$ is

$$F(x) = \begin{cases} 0 & if \quad x < 0 \\ 10x^6 + 15x^4 - 24x^5 & if \quad 0 < x < 1 \\ 1 & if \quad x > 1 \end{cases}$$

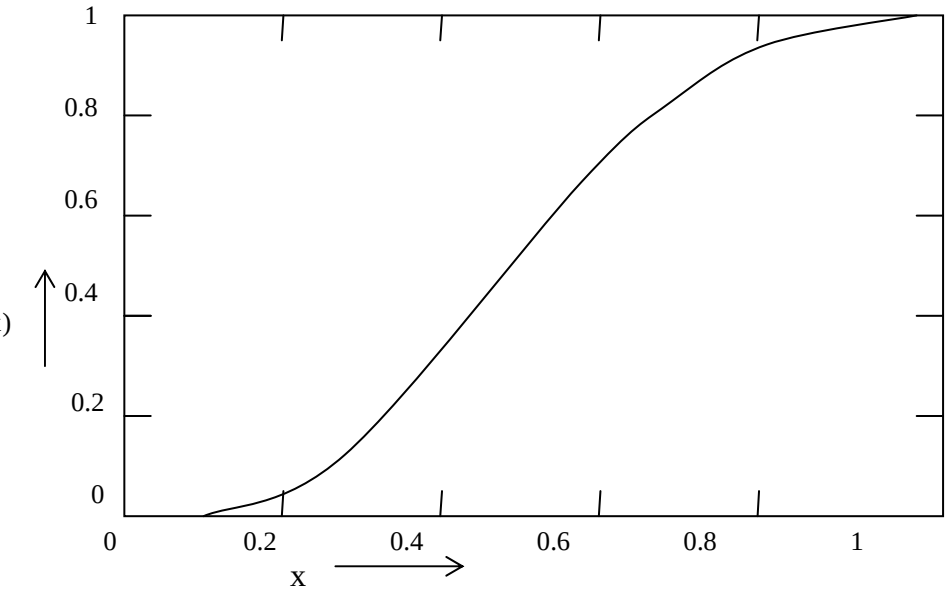
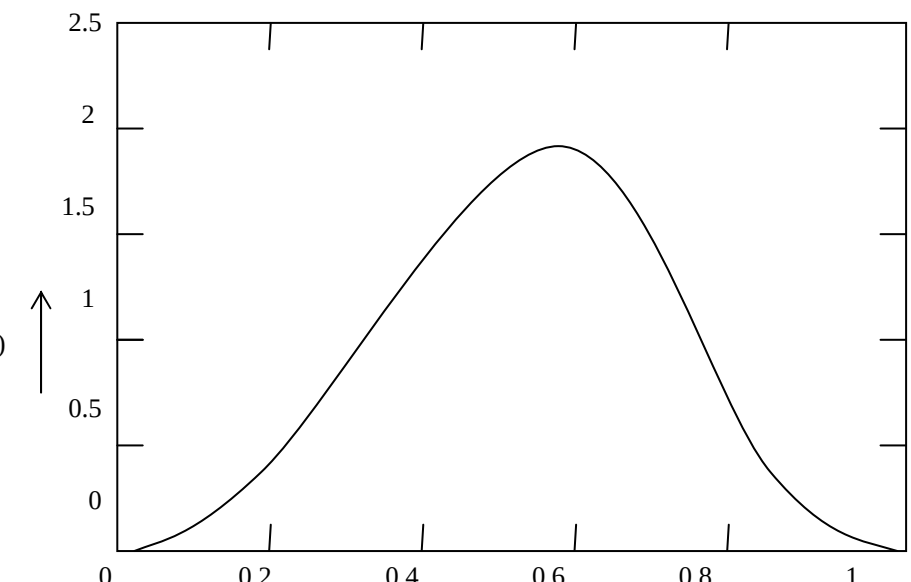


Figure 4 : Distribution function of the density function $f(x)$



From the distribution function $F(x)$ is provided in figure 4 .It is clear that the distribution function $F(x)$ is a strictly increasing function of x in [0, 1]. Therefore $F^{-1}(x)$ exists, but unfortunately to find $F^{-1}(x)$ we need to solve a six degree polynomial, which can not be obtained analytically. We need to solve numerically only. Therefore, we can not generate random deviate from the density function $f(x)$ using the inversion method. But we will be able to generate random deviate from $f(x)$ using the acceptance-rejection method.

First let us look at the graph of the density function $f(x)$. It is presented in the Figure 5. From the Figure 5 it is clear that $f(x)$ is an unimodal function with a unique maximum. The maximum can be easily obtained by the standard differential calculus, that is by setting $\frac{df(x)}{dx} = 0$. We see that the maximum of $f(x)$ occurs at $x = 0.6$ and the maximum value at 0.6, that is $f(0.6) = 2.0736$. Therefore, if we define

$$g(x) = \begin{cases} 2.0736 & if \quad 0 < x < 1 \\ 0 & \text{otherwise.} \end{cases}$$

then clearly $f(x) \le g(x)$ for all x. Now to calculate $h(x)$, first we need to calculate c. Since,

$c = \int_0^1 2.0735 = 2.0736$, therefore,

$$h(\underline{x}) = \begin{cases} \frac{2.0736}{c} = 1 & if \quad 0 < x < 1 \\ 0 & \text{otherwise.} \end{cases}$$

It is immediate that $h(x)$ is just the U(0,1) density function. Now the algorithm takes the following form.

- Step 1: Generate Y from $U(0, 1)$
- Step 2: Generate U from $U(0,1)$ which is independent of Y.
- Step 3: If $U \le \frac{60Y^3(1-Y)^2}{2.0736}$ then return with $X = Y$,otherwise go back to Step 1.

In this case X has the desired density function $f(x)$.

Check your progress 3

E1 use acceptance-rejection method to generate a random deviate from gamma density function, . The gamma density function with the shape parameter α can be written as

$$f(x) = \begin{cases} \frac{1}{\Gamma(\alpha)} x^{\alpha-1} e^{-x} & if \quad x > 0 \\ 0 & \text{otherwise} \end{cases}$$

2. 6 SUMMARY

In this unit we have discussed the meaning of pseudo random number generation and along with that we have described uniform random number generator and arbitrary random number generator. Under Uniform random number generator case we have emphasized on **LCG (Linear Congruential Generator)** and some objections related to LCG. Actually LCGs provides an algorithm how to generate uniform random number between (0,1). It can be simply described as follows.

A sequence of integers $Z_1, Z_2, \ldots$ is defined by the recursive formula

$$Z_i = (aZ_{i-1} + c) \pmod{m}, \qquad (1)$$

where *m*, the modulus, *a*, the multiplier, *c*, the increment and Z_0, the seed or the initial value, are all non-negative integers

Then we have discussed the concept , algorithm and application of **Inverse transforms** for random number generation . In brief Suppose we want to generate a random variate *X* that has a continuous and strictly increasing distribution function *F*, when $0 < F(x) < 1$, i.e., whenever $x_1 < x_2$ then $F(x_1) < F(x_2)$. Let F^{-1} denote the inverse of the function *F*. Then an algorithm for generating a random variate of *X* having distribution function *F* is as follow .

ALGORITHM

- Step 1: Generate U_1 from U_1 (0,1)
- Step 2: Return $X_1 = F^{-1}(U_1)$.

The inverse-transform method can be used also when the random variable X is discrete.
In this case the distribution function is

$$F(x) = P(X \le x) = \sum_{x_i \le x} p(x_i),$$

Where $p(x_i)$ is the probability mass function of *X, i.e.,*

$$p(x_i) = P(X = x_i).$$

We can assume that *X* can take values $x_1, x_2, \ldots$, where $x_1 < x_2 < \ldots$. Then the algorithm is as follows:

ALGORITHM

- Step 1: Generate *U* from *U* (0,1)
- Step 2: Determine the smallest positive integer *I* such that $U \le F(x_1)$ and return $X = x_1$. The following figure 2 illustrates the method. In that case we generate $X=x_4$

Note:

- the inverse transformation method to generate random number from different non-uniform distributions. Note that apparently, the inverse transformation method seems to be the most general method to generate random deviate from any distribution function functions. In fact it can be used provided the distribution function can be written in an explicit form

Finally we had discussed the **Acceptance Rejection method** of random number generation, this method is quite important because ,when the distribution function can not be computed analytically and similarly it's inverse also. Then in such cases we can not apply the inverse transformation method to generate the corresponding random deviates. The acceptance-rejection method can be used quite effectively to generate these random deviates.

In brief ,Suppose we have a density function $f(x)$ and we want to generate a random deviate from the density function $f(x)$. The distribution function of $f(x)$ can not be expressed in explicit form. The acceptance-rejection method requires that we specify a function $g(x)$ that majorizes the function $f(x)$, that is, $f(x) \le g(x)$ for all *x*. Naturally, $g(x)$ will not be a density function always, since

$$c = \int_{\infty}^{\infty} g(x)dx \ge \int_{\infty}^{\infty} f(x) = 1,$$

but the function $h(x) = \frac{1}{c} g(x)$ is clearly a density function provided $c < \infty$. Now for any given $f(x)$, we choose the function $g(x)$, such that $c < \infty$ and it is possible to generate random deviate from $g(x)$ by a inverse transformation method. Then we can generate the random deviate *X* from $f(x)$ as follows:

ALGORITHM

- Step 1: Generate *Y* having density function $g(x)$.
- Step 2: Generate *U* from *U* (0,1) which is independent of *Y*.
- Step 3: If $U \le \frac{f(Y)}{g(Y)}$, $X = Y$, otherwise go back to Step 1 and try again.

Note that the algorithm is looping back to Step 1 until we generate a pair (*Y, U*) pairs in Steps 1 and 2 for which $U \le \frac{f(Y)}{g(Y)}$, when we accept the value *Y* for *X*. Theoretically it can be shown that the random deviate *X* generated by the above algorithm has indeed the probability density function $f(x)$. Since it is not very easy to prove the result we do not provide it.

2. 7 SOLUTIONS

Check your progress 1

E1 A pseudo-random number generation is the methodology to develop algorithms and programs that can be used in, probability and statistics applications when large quantities of random digits are needed. Most of these programs produce endless strings of single-digit numbers, usually in base 10, known as the decimal system. When large samples of pseudo-random numbers are taken, each of the 10 digits in the set {0,1,2,3,4,5,6,7,8,9} occurs with equal frequency, even though they are not evenly distributed in the sequence.

Many algorithms have been developed in an attempt to produce truly random sequences of numbers, endless strings of digits in which it is theoretically impossible to predict the next digit in the sequence based on the digits up to a given point. But the very existence of the algorithm, no matter how sophisticated, means that the next digit can be predicted! This has given rise to the term pseudo-random for such machine-generated strings of digits. They are equivalent to random-number sequences for most applications, but they are not truly random according to the rigorous definition.

There are several methods available to generate uniform random numbers. But currently the most popular one is the linear congruential generator (LCG). Most of the existing software's today use this LCGs proposed by Lehmer in the early 50's.

Check your progress 2

E1 To find F^{-1}, we set $u = F(x)$ and solve for x to obtain

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

Therefore, to generate random variate X from exponential distribution with mean 1, first generate U from a U (0,1) and then let X = - In (1 – U). Therefore X will have exponential distribution with mean 1. Since U and 1 – U have the same U (0, 1) distribution, we can also use X = In U. This saves a subtraction.

E2 Note that $\sum_{i=1}^{\infty} \frac{1}{2^i}$ = 1, therefore p_i denotes the probability mass function of a discrete random variable. The corresponding distribution function of the discrete random variable X can be written as

$$F(x) = \begin{cases} 0 & if \quad x < 1 \\ \sum_{i=1}^{m} \frac{1}{2^i} & if \quad m \le x < m+1, \end{cases}$$

where m is any positive integer. Now to generate a random deviate from the random variable X, first draw a random sample U from U (0, 1), since $0 \le U \le 1$, there exits a positive integer m such that $\sum_{i=1}^{m-1} \frac{1}{2^i} \le \mathrm{U} < \sum_{i=1}^{m} \frac{1}{2^i}$, where $\sum_{i=1}^{0} \frac{1}{2^i} = 0$, then $X = m$,

Check your progress 3

E1 Our problem is to generate random deviate from $f(x)$ for a given $0 < \alpha < 1$. Note that we can not use the acceptance-rejection method in this case. It is easily observed if we take

$$g(x) = \begin{cases} 0 & if \quad x \le 0 \\ \frac{x^{\alpha-1}}{\Gamma(\alpha)} & if \quad 0 < x < 1 \\ \frac{e^{-x}}{\Gamma(\alpha)} & if \quad x > 1, \end{cases}$$

then $f(x) \le g(x)$ for all x. In this case

$$c = \int_{\infty}^{\infty} g(x) = \int_0^1 \frac{x^{\alpha-1}}{\Gamma(\alpha)} dx + \int_1^{\infty} \frac{e^{-x}}{\Gamma(\alpha)} dx = \frac{1}{\Gamma(\alpha)} \left[\frac{(e+a)}{ae} \right].$$

Therefore, $h(x) = \frac{1}{c}\, g(x)$ is

$$h(x) = \begin{cases} 0 & if \quad x \le 0 \\ \dfrac{\alpha x^{\alpha-1}}{b} & if \quad 0 \le x \le 1 \\ \dfrac{\alpha e^{-x}}{b} & if \quad x > 1, \end{cases}$$

where $b = \frac{e+a}{e}$. The distribution function H (x) corresponds to the density function $h(x)$ is

$$H(x) = \int_0^x h(y)\, dy = \begin{cases} \dfrac{x^{\alpha}}{b} & if \quad 0 \le x \le 1 \\ 1 - \dfrac{\alpha e^{-x}}{b} & if \quad x > 1, \end{cases}$$

Which can be easily inverted as

$$H^{-1}(u) = \begin{cases} (bu)^{\frac{1}{\alpha}} & if \quad u \le \dfrac{1}{b} \\ -\ln \dfrac{b(1-u)}{\alpha} & if \quad u > \dfrac{1}{b} \end{cases}$$

Therefore, it is possible to generate random deviate from the density function $h(x)$ using the simple inversion method. Generation of a random deviate Y from the density function $h(x)$ can be performed as follows. First generate U_1 from U (0, 1), if $U_1 \le \frac{1}{b}$ we set $Y = (bU_1)^{\frac{1}{\alpha}}$, in this case $Y \le 1$. Otherwise $Y = -\ln \frac{b(1-U_1)}{\alpha}$ and in this case $Y > 1$. Also note that

$$\frac{f(Y)}{g(Y)} = \begin{cases} e^{-Y} & if \quad 0 \le Y \le 1 \\ Y^{\alpha-1} & if \quad Y > 1 \end{cases}$$

Now the algorithm to generate random deviate from a gamma density function with the shape parameter α, for $0 < \alpha < 1$ takes the following form:

- Step 1: Generate U_1 from U (0,1) and let $P = bU_1$. If $P > 1$, go to Step 3 otherwise proceed to Step 2.

- Step 2: Let $Y = P^{\frac{1}{\alpha}}$ and generate U_2 from U (0, 1). If $U_2 \le e^{-Y}$, return $X = Y$.
  Otherwise go back to Step 1.

- Step 3: Let $Y = -\ln \frac{(b-P)}{\alpha}$ and generate U_2 from U (0, 1). If $U_2 \le Y^{\alpha-1}$, return $X = Y$, otherwise go back to Step 1.