Random Number Generation

Norm Matloff

February 21, 2006

Contents

1 Uniform Random Number Generation

The basic building block for generating random numbers from various distributions is a generator of uniform random numbers on the interval (0,1), i.e. from the distribution U(0,1).[1] How is this accomplished?

Due to the finite precision of computers, we cannot generate a true continuous random variate. However, we will generate a discrete random variate which behaves very close to U(0,1).

There is a very rich literature on the generation of random integers, commonly called **pseudorandom numbers** because they are actually deterministic. Pseudorandom numbers can be divided by their upper bound to generate U(0,1) variates. Here is an example (in pseudocode):

```
1 int U01()
2 constant C = 25173
3 constant D = 13849
4 constant M = 32768
5 static Seed
6 Seed = (C*Seed + D) % M
7 return Seed/((float) M)
```

The name **Seed** stems from the fact that most random number libraries ask the user to specify what they call the **seed value**. You can now see what that means. The value the user gives "seeds" the sequence of random numbers that are generated. Note that **Seed** is declared as **static**, so that it retains its value between calls.

This certainly will give us values in (0,1), and intuitively it is plausible that they are "random." But is this a "good" approximation to a generator of U(0,1) variates? What does "good" really mean?

Here is what we would like to happen. Suppose we call the function again and again, with X_i being the value returned by the i^{th} call. Then ideally the X_i should be independent, which means we would have

- For any $0 < r < s < 1$,

$$\lim_{n \to \infty} \frac{C(r,s,n)}{n} = s - r \tag{1}$$

  where C(r,s,n) is a count of the number of X_i which fall into (r,s), i = 1,...,n.

- For any $0 < r < s < 1$, $0 < u < v < 1$ and integer $k > 0$,

$$\lim_{n \to \infty} \frac{D(r,s,u,v,n,k)}{n} = (s - r)(v - u) \tag{2}$$

  where D(r,s,u,v,n,k) is the count of the number of i for which $r < X_i < s$ and $u < X_{i+k} < v$, i = 1,...,n.

- The third- and higher-dimensional versions of (2) hold.

[1] Since a random variable which is uniformly distributed on some interval places 0 mass on any particular point, it doesn't matter whether we talk here of (0,1), [0,1], etc.

Equation (1) is saying that the X_i are uniformly distributed on (0,1), and (2) and its third- and higher-dimensional versions say that the X_i are independent. How close can we come to fully satisfying these conditions?

To satisfy (1), the algorithm needs to have the property that **Seed** can take on all values in the set $\{0, 1, 2, ..., M-1\}$, without repetition (until they are all hit). It can be shown that this condition will hold if all the following are true:

- **D** and **M** are relatively prime
- **C-1** is divisible by every prime factor of **M**
- if **M** is divisible by 4, then so is **C-1**

Typically **M** is chosen according to the word size of the machine. On today's 32-bit machines **M** might be 2^{32}, or about 4 billion. This makes it easy to do the **mod M** operation.

However, determining which values of **C** and **D** give approximate independence is something that requires experimental investigation. Again, there is a rich literature on this, but we will not pursue it here. You can assume, though, that any reputable library package has chosen values which work well.[2]

2 Generating Random Numbers from Continuous Distributions

There are various methods to generate continuous random variates. We'll introduce some of them here.

2.1 The Inverse Transformation Method

2.1.1 General Description of the Method

Suppose we wish to generate random numbers having density h. Let H denote the corresponding cumulative distribution function, and let $G = H^{-1}$ (inverse in the same sense as square and square root operations are inverses of each other). Set X = G(U), where U has a U(0,1) distribution. Then

$$\begin{aligned} F_X(t) &= P(X \leq t) \\ &= P(G(U) \leq t) \\ &= P(U \leq G^{-1}(t)) \\ &= P(U \leq H(t)) \\ &= H(t) \end{aligned} \tag{3}$$

In other words, X has density h, as desired.

[2] And of course you can do your own experimental investigation on it if you wish.

2.1.2 Example: The Exponential Family

For example, suppose we wish to generate exponential random variables with parameter λ. In this case,

$$H(t) = \int_0^t \lambda e^{-\lambda s}\, ds = 1 - e^{-\lambda t} \tag{4}$$

Writing u = H(t) and solving for t, we have

$$G(u) = -\frac{1}{\lambda} \ln(1-u) \tag{5}$$

So, pseudocode for a function to generate such random variables would be

```
float Expon(float Lambda)
return -1.0/Lambda * log(1-U01())
```

If a random variable Y has a U(0,1) distribution then so does 1-Y. We might be tempted to exploit this fact to save a step above, using **U01()** instead of **1-U01()**. But if U01() returns 0, we are then faced with computing log(0), which is undefined. (U01() never returns 1, as formulated in Section 1.)

2.2 The Acceptance/Rejection Method

2.2.1 General Description of the Method

Suppose again we wish to generate random numbers having density h. With h being the exponential density above, H^{-1} was easy to find, but in many other cases it is intractable. Here is another method we can use:

We think of some other density g such that

- We know how to generate random variables with density g.
- There is some c such that $h(x) \leq cg(x)$ for all x.

Then it can be shown that the following will generate random variates with density h:

```
while true:
   generate Y from g
   generate U from U(0,1)
   if U <= h(Y)/(c*g(Y)) return Y
```

2.2.2 Example

As an example, consider the u-shaped density $h(z) = 12(z-0.5)^2$ on (0,1). Take our g to be the U(0,1) density, and take c to be the maximum value of h, which is 3. Our code would then be

```
while true:
   generate U1 from U(0,1)
   generate U2 from U(0,1)
   if U2 <= 4*(U1-0.5)**2 return U1
```

2.3 *Ad Hoc* Methods

In some cases, we can see a good way to generate random numbers from a given distribution by thinking of the properties of the distribution.

2.3.1 Example: The Erlang Family

For example, consider the Erlang family of distributions, whose densities are given by

$$\frac{1}{(r-1)!}\lambda^r t^{r-1} e^{-\lambda t},\ t > 0 \tag{6}$$

Recall that this happens to be the density of the sum of r independent random variables which are exponentially distributed with rate parameter λ. This then suggests an easy way to generate random numbers from an Erlang distribution:

```
1  Sum = 0
2  for i = 1,...,r
3     Sum = Sum + expon(lambda)
4  return Sum
```

where of course we mean the function expon() to be a function that generates exponentially distributed random variables, such as in Section 2.1.2.

2.3.2 Example: The Normal Family

Note first that all we need is a method to generate normal random variables of mean 0 and variance 1. If we need random variables of mean μ and standard deviation σ, we simply generate N(0,1) variables, multiply by σ and adding μ.

So, how do we generate N(0,1) variables? The inverse transformation method is infeasible, since the normal density cannot be integrated in closed form. We could use the acceptance/rejection method, with g being the exponential density with rate 1. After doing a little calculus, we would find that we could take $c = \sqrt{2e/\pi}$.

However, a better way exists. It can be shown that this works:

```
1  static n = 0
2  static Y
3  if n == 0
4     generate U1 and U2 from U(0,1)
5     R = sqrt(-2 log(U1))
6     T = 2 pi U2
7     X = R cos(T)
8     Y = R sin(T)
9     n = 1
10    return X
11 else
12    n = 0
13    return Y
```

3 Generating Random Numbers from Discrete Distributions

3.1 The Inverse Transformation Method

The method in Section 2.1 works in the discrete case too. Suppose we wish to generate random variates X which take on the values 0,1,2,... Let

$$q(i) = P(X \leq i) = \sum_{k=0}^{i} p_X(i) \tag{7}$$

Then our algorithm is

```
1  generate U from U(0,1)
2  I = 0
3  while true
4     if U < q(I) return I
5     I = I + 1
```

3.2 *Ad Hoc* Methods

Again, for certain specific distributions, we can sometimes make use of special properties that we know about those distributions.

3.2.1 The Binomial Family

This one is obvious:

```
1  Count = 0
2  for I = 1,..,n
3     generate U from U(0,1)
4     if U < p Count = Count + 1
5  return Count
```

3.2.2 The Geometric Family

Again, obvious:

```
1  Count = 0
2  while true:
3     Count = Count + 1
4     generate U from U(0,1)
5     if U < p return Count
```

3.2.3 The Poisson Family

Recall that if events, say arrivals to a queue, have interevent times which are independent exponentially distributed random variables with rate λ, then the number of events N(t) up to time t has a Poisson distribution with parameter λt:

$$P(N(t) = k) = \frac{e^{-\lambda t}(\lambda t)^k}{k!}, k = 0, 1, 2, ... \tag{8}$$

So, if we wish to generate random variates having a Poisson distribution with parameter α, we can do the following:

```
1  Count = 0
2  Sum = 0
3  while 1
4     Count = Count + 1
5     Sum = Sum + expon(alpha)
6     if Sum > 1 return Count-1
```

where again expon(alpha) means generating an exponentially distributed random variable, in this case with rate α.