

7.2 Transformation Method: Exponential and Normal Deviates

In the previous section, we learned how to generate random deviates with a uniform probability distribution, so that the probability of generating a number between x and $x + dx$, denoted $p(x)dx$, is given by

$$p(x)dx = \begin{cases} dx & 0 < x < 1 \\ 0 & \text{otherwise} \end{cases} \quad (7.2.1)$$

The probability distribution $p(x)$ is of course normalized, so that

$$\int_{-\infty}^{\infty} p(x)dx = 1 \quad (7.2.2)$$

Now suppose that we generate a uniform deviate x and then take some prescribed function of it, $y(x)$. The probability distribution of y , denoted $p(y)dy$, is determined by the fundamental transformation law of probabilities, which is simply

$$|p(y)dy| = |p(x)dx| \quad (7.2.3)$$

or

$$p(y) = p(x) \left| \frac{dx}{dy} \right| \quad (7.2.4)$$

Exponential Deviates

As an example, suppose that $y(x) \equiv -\ln(x)$, and that $p(x)$ is as given by equation (7.2.1) for a uniform deviate. Then

$$p(y)dy = \left| \frac{dx}{dy} \right| dy = e^{-y} dy \quad (7.2.5)$$

which is distributed exponentially. This exponential distribution occurs frequently in real problems, usually as the distribution of waiting times between independent Poisson-random events, for example the radioactive decay of nuclei. You can also easily see (from 7.2.4) that the quantity y/λ has the probability distribution $\lambda e^{-\lambda y}$.

So we have

```
#include <math.h>
```

```
float expdev(long *idum)
```

Returns an exponentially distributed, positive, random deviate of unit mean, using `ran1(idum)` as the source of uniform deviates.

```
{
    float ran1(long *idum);
    float dum;

    do
        dum=ran1(idum);
    while (dum == 0.0);
    return -log(dum);
}
```

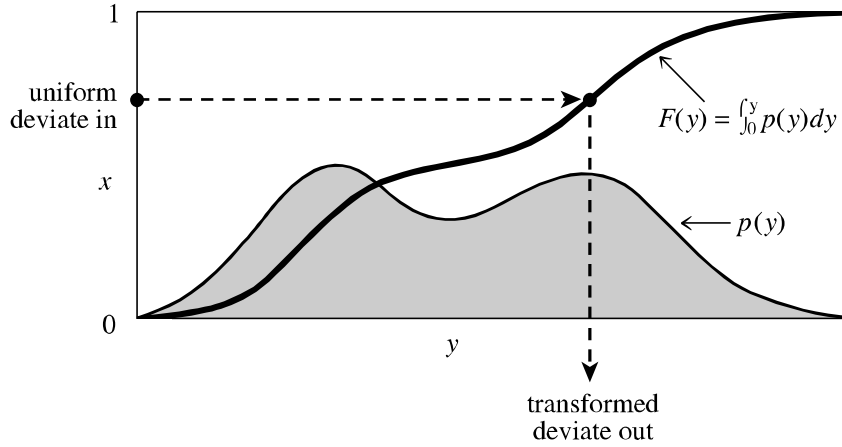


Figure 7.2.1. Transformation method for generating a random deviate y from a known probability distribution $p(y)$. The indefinite integral of $p(y)$ must be known and invertible. A uniform deviate x is chosen between 0 and 1. Its corresponding y on the definite-integral curve is the desired deviate.

Let's see what is involved in using the above *transformation method* to generate some arbitrary desired distribution of y 's, say one with $p(y) = f(y)$ for some positive function f whose integral is 1. (See Figure 7.2.1.) According to (7.2.4), we need to solve the differential equation

$$\frac{dx}{dy} = f(y) \quad (7.2.6)$$

But the solution of this is just $x = F(y)$, where $F(y)$ is the indefinite integral of $f(y)$. The desired transformation which takes a uniform deviate into one distributed as $f(y)$ is therefore

$$y(x) = F^{-1}(x) \quad (7.2.7)$$

where F^{-1} is the inverse function to F . Whether (7.2.7) is feasible to implement depends on whether the *inverse function of the integral of $f(y)$* is itself feasible to compute, either analytically or numerically. Sometimes it is, and sometimes it isn't.

Incidentally, (7.2.7) has an immediate geometric interpretation: Since $F(y)$ is the area under the probability curve to the left of y , (7.2.7) is just the prescription: choose a uniform random x , then find the value y that has that fraction x of probability area to its left, and return the value y .

Normal (Gaussian) Deviates

Transformation methods generalize to more than one dimension. If x_1, x_2, \dots are random deviates with a *joint* probability distribution $p(x_1, x_2, \dots)$ $dx_1 dx_2 \dots$, and if y_1, y_2, \dots are each functions of all the x 's (same number of y 's as x 's), then the joint probability distribution of the y 's is

$$p(y_1, y_2, \dots) dy_1 dy_2 \dots = p(x_1, x_2, \dots) \left| \frac{\partial(x_1, x_2, \dots)}{\partial(y_1, y_2, \dots)} \right| dy_1 dy_2 \dots \quad (7.2.8)$$

where $|\partial(\quad)/\partial(\quad)|$ is the Jacobian determinant of the x 's with respect to the y 's (or reciprocal of the Jacobian determinant of the y 's with respect to the x 's).

An important example of the use of (7.2.8) is the *Box-Muller* method for generating random deviates with a normal (Gaussian) distribution,

$$p(y)dy = \frac{1}{\sqrt{2\pi}} e^{-y^2/2} dy \quad (7.2.9)$$

Consider the transformation between two uniform deviates on (0,1), x_1, x_2 , and two quantities y_1, y_2 ,

$$\begin{aligned} y_1 &= \sqrt{-2 \ln x_1} \cos 2\pi x_2 \\ y_2 &= \sqrt{-2 \ln x_1} \sin 2\pi x_2 \end{aligned} \quad (7.2.10)$$

Equivalently we can write

$$\begin{aligned} x_1 &= \exp \left[-\frac{1}{2}(y_1^2 + y_2^2) \right] \\ x_2 &= \frac{1}{2\pi} \arctan \frac{y_2}{y_1} \end{aligned} \quad (7.2.11)$$

Now the Jacobian determinant can readily be calculated (try it!):

$$\frac{\partial(x_1, x_2)}{\partial(y_1, y_2)} = \begin{vmatrix} \frac{\partial x_1}{\partial y_1} & \frac{\partial x_1}{\partial y_2} \\ \frac{\partial x_2}{\partial y_1} & \frac{\partial x_2}{\partial y_2} \end{vmatrix} = - \left[\frac{1}{\sqrt{2\pi}} e^{-y_1^2/2} \right] \left[\frac{1}{\sqrt{2\pi}} e^{-y_2^2/2} \right] \quad (7.2.12)$$

Since this is the product of a function of y_2 alone and a function of y_1 alone, we see that each y is independently distributed according to the normal distribution (7.2.9).

One further trick is useful in applying (7.2.10). Suppose that, instead of picking uniform deviates x_1 and x_2 in the unit square, we instead pick v_1 and v_2 as the ordinate and abscissa of a random point inside the unit circle around the origin. Then the sum of their squares, $R^2 \equiv v_1^2 + v_2^2$ is a uniform deviate, which can be used for x_1 , while the angle that (v_1, v_2) defines with respect to the v_1 axis can serve as the random angle $2\pi x_2$. What's the advantage? It's that the cosine and sine in (7.2.10) can now be written as $v_1/\sqrt{R^2}$ and $v_2/\sqrt{R^2}$, obviating the trigonometric function calls!

We thus have

```
#include <math.h>
```

```
float gasdev(long *idum)
```

Returns a normally distributed deviate with zero mean and unit variance, using `ran1(idum)` as the source of uniform deviates.

```
{
```

```
    float ran1(long *idum);
    static int iset=0;
    static float gset;
    float fac,rsq,v1,v2;
```

```
    if (*idum < 0) iset=0;
```

```
    if (iset == 0) {
```

```
        do {
```

```
            v1=2.0*ran1(idum)-1.0;
```

```
            v2=2.0*ran1(idum)-1.0;
```

```
            rsq=v1*v1+v2*v2;
```

Reinitialize.

We don't have an extra deviate handy, so

pick two uniform numbers in the square extending from -1 to +1 in each direction, see if they are in the unit circle,

```

    } while (rsq >= 1.0 || rsq == 0.0);      and if they are not, try again.
    fac=sqrt(-2.0*log(rsq)/rsq);
    Now make the Box-Muller transformation to get two normal deviates. Return one and
    save the other for next time.
    gset=v1*fac;
    iset=1;                                Set flag.
    return v2*fac;
} else {                                  We have an extra deviate handy,
    iset=0;                                so unset the flag,
    return gset;                            and return it.
}
}

```

See Devroye [1] and Bratley [2] for many additional algorithms.

CITED REFERENCES AND FURTHER READING:

- Devroye, L. 1986, *Non-Uniform Random Variate Generation* (New York: Springer-Verlag), §9.1. [1]
- Bratley, P., Fox, B.L., and Schrage, E.L. 1983, *A Guide to Simulation* (New York: Springer-Verlag). [2]
- Knuth, D.E. 1981, *Seminumerical Algorithms*, 2nd ed., vol. 2 of *The Art of Computer Programming* (Reading, MA: Addison-Wesley), pp. 116ff.