Marsaglia polar method

From Wikipedia, the free encyclopedia

The **polar method** (attributed to George Marsaglia, 1964[1]) is a pseudo-random number sampling method for generating a pair of independent standard normal random variables. While it is superior to the Box–Muller transform, the Ziggurat algorithm is even more efficient.[2]

Standard normal random variables are frequently used in computer science, computational statistics, and in particular, in applications of the Monte Carlo method.

The polar method works by choosing random points (x, y) in the square $-1 < x < 1$, $-1 < y < 1$ until

$$s = x^2 + y^2 < 1,$$

and then returning the required pair of normal random variables as

$$x\sqrt{\frac{-2\ln(s)}{s}},\quad y\sqrt{\frac{-2\ln(s)}{s}}.$$

Contents

Theoretical basis

The underlying theory may be summarized as follows:

If u is uniformly distributed in the interval $0 \le u < 1$, then the point $(\cos(2\pi u), \sin(2\pi u))$ is uniformly distributed on the unit circumference $x^2 + y^2 = 1$, and multiplying that point by an independent random variable ρ whose distribution is

$$\Pr(\rho < a) = \int_0^a re^{-r^2/2}\,dr$$

will produce a point

$$(\rho\cos(2\pi u), \rho\sin(2\pi u))$$

whose coordinates are jointly distributed as two independent standard normal random variables.

History

This idea dates back to Laplace, whom Gauss credits with finding the above

$$I = \int_{-\infty}^{\infty} e^{-x^2/2}\,dx$$

by taking the square root of

$$I^2 = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} e^{-(x^2+y^2)/2}\,dx\,dy = \int_0^{2\pi}\int_0^{\infty} re^{-r^2/2}\,dr\,d\theta.$$

The transformation to polar coordinates makes evident that θ is uniformly distributed (constant density) from 0 to 2π, and that the radial distance *r* has density

$$re^{-r^2/2}.$$

(r^2 has the appropriate chi square distribution.)

This method of producing a pair of independent standard normal variates by radially projecting a random point on the unit circumference to a distance given by the square root of a chi-square-2 variate is called the polar method for generating a pair of normal random variables,

Practical considerations

A direct application of this idea,

$$x = \sqrt{-2\ln(u_1)}\cos(2\pi u_2), \quad y = \sqrt{-2\ln(u_1)}\sin(2\pi u_2)$$

is called the Box Muller transform, in which the chi variate is usually generated as

$$\sqrt{-2\ln(u_1)};$$

but that transform requires logarithm, square root, sine and cosine functions. On some processors, the cosine and sine of the same argument can be calculated in parallel using a single instruction.[3] Notably for Intel based machines, one can use fsincos assembler instruction or the expi instruction (available e.g. in D), to calculate complex

$$\operatorname{expi}(z) = e^{iz} = \cos(z) + i\sin(z),$$

and just separate the real and imaginary parts.

The polar method, in which a random point (x, y) inside the unit circle is projected onto the unit circumference by setting $s = x^2 + y^2$ and forming the point

$$\left(\frac{x}{\sqrt{s}}, \frac{y}{\sqrt{s}}\right),$$

is a faster procedure. Some researchers argue that the conditional if instruction (for rejecting a point outside of the unit circle), can make programs slower on modern processors equipped with pipelining and branch prediction. Also this procedure requires about 21% more evaluations of the underlying random number generator (only $\pi/4 \approx 79\%$ of generated points lie inside of unit circle).

That random point on the circumference is then radially projected the required random distance by means of

$$\sqrt{-2\ln(s)},$$

using the same s because that s is independent of the random point on the circumference and is itself uniformly distributed from 0 to 1.

Implementation

Simple implementation in Java using the mean and standard deviation:

```java
private static double spare;
private static boolean isSpareReady = false;

public static synchronized double getGaussian(double mean, double stdDev) {
    if (isSpareReady) {
        isSpareReady = false;
        return spare * stdDev + mean;
    } else {
        double u, v, s;
        do {
            u = Math.random() * 2 - 1;
            v = Math.random() * 2 - 1;
            s = u * u + v * v;
        } while (s >= 1 || s == 0);
        double mul = Math.sqrt(-2.0 * Math.log(s) / s);
        spare = v * mul;
        isSpareReady = true;
        return mean + stdDev * u * mul;
    }
}
```

An implementation in C++ using the variance:

```
double generateGaussianNoise(const double &variance)
{
    static bool hasSpare = false;
    static double spare;

    if(hasSpare)
    {
        hasSpare = false;
        return variance * spare;
    }

    hasSpare = true;
    static qreal u, v, s;
    do
    {
        u = (rand() / ((double) RAND_MAX)) * 2 - 1;
        v = (rand() / ((double) RAND_MAX)) * 2 - 1;
        s = u * u + v * v;
    }
    while(s >= 1 || s == 0);

    s = sqrt(-2.0 * log(s) / s);
    spare = v * s;
    return variance * u * s;
}
```

References

1. ^ A convenient method for generating normal variables, G. Marsaglia and T. A. Bray, SIAM Rev. 6, 260–264, 1964 (http://www.jstor.org/stable/2027592)
2. ^ http://doi.acm.org/10.1145/1287620.1287622 Gaussian Random Number Generators, D. Thomas and W. Luk and P. Leong and J. Villasenor, ACM Computing Surveys, Vol. 39(4), Article 11, 2007, doi:10.1145/1287620.1287622 (http://dx.doi.org/10.1145%2F1287620.1287622)
3. ^ Kanter, David. "Intel's Ivy Bridge Graphics Architecture" (http://www.realworldtech.com/ivy-bridge-gpu/5/). *Real World Tech*. Retrieved 8 April 2013.