Three Different Algorithms for Generating Uniformly Distributed Random Points on the N-Sphere

Jan Poland

Oct 24, 2000

Abstract

We present and compare three different approaches to generate random points on the N-sphere: A simple Monte Carlo algorithm, a coordinate-by-coordinate strategy and a method based on the rotation invariance the normal distribution. The latter algorithm is the fastest.

1 Introduction

Computer scientists sometimes encounter the problem of generating random points on the N-dimensional unit sphere $S_N = \{x \in \mathbb{R}^{N+1} : \|x\|_2 = 1\}$. In most cases, the random points should be distributed *uniformly*. For this task, there exists a nice little algorithm already mentioned in Knuth ([3]), that is based on the fact that the N-dimensional normal distribution is invariant under rotation. This algorithm is indubitably the most efficient, in particular in high dimensions. But if a special non-uniform distribution is requested, it may be profitable to know other approaches such as a coordinate-by-coordinate strategy or a simple rejection method.

When N is small, each algorithm is sufficiently fast, and the problem is not very interesting. For the 1-sphere, i.e. the unit circle in $\mathbb{R}^2$, $U = (\cos(\alpha), \sin(\alpha))$ with α uniformly distributed in $[0, 2\pi]$ does the job. Here, we will concentrate on the high dimensional case. Applications are for example in the initialization of a counterpropagation neural network (see [7]) or a variant of generalized continuous recombination in an evolution strategy (cf. [4]).

Other discussions of our problem can be found in [6] or [5].

2 A Monte-Carlo algorithm

An immediate algorithm is the following. Take a random point in $[-1,1]^{N+1}$ and project it onto the unit sphere . This results in a non-uniform distribution, therefore one extra step is necessary. Generate a random point in $X \in [-1,1]^{N+1}$ and reject it if it is outside the unit ball, i.e. $\|X\| > 1$. If not, $U = \|X\|^{-1} \cdot X$ gives a random point on S_N (neglecting the unlikely case that $X = 0$), and produces a uniform distribution on the sphere.

This algorithm is often referred to as the rejection method (see [6]). The major drawback of the algorithm is it's expected running time: It grows worse than exponentially in N. This is due to the fact that the volume of the N-dimesional unit ball

$$V(N) = \frac{\pi^{\frac{N}{2}}}{\Gamma(1+\frac{N}{2})}$$

more than exponentially decreases as $N \to \infty$, where $\Gamma(\cdot)$ is the gamma function

$$\Gamma(x) = \int_0^\infty e^{-t} \cdot t^{x-1}\, dt.$$

3 The coordinate approach

In high dimensions, the distribution of *each single coordinate* of a uniformly distributed random point U on the N-sphere becomes quite complicated.

Suppose we are given a random vector U uniformly distributed on S_N. Consider the projection U_1 of U to its first coordinate, lets say the x-coordinate. Then the density function $f_1(x)$ of U_1 is a bounded function from $[-1,1]$ into $\mathbb{R}^+$. We now want to state the f_1 explicitely.

The following computations will need the incomplete beta function

$$B_x(a,b) \;=\; \int_0^x t^{a-1} \cdot (1-t)^{b-1}\, dt$$

and the beta function $B(a,b) = B_1(a,b)$.

To determine f_1, take a point $u \in S_N$ and consider a small move along the first coordinate to another point $\tilde{u} \in S_N$. We observe that the move approximately covers the distance

$$\|\tilde{u} - u\| \approx |\arcsin(\tilde{x}) - \arcsin(x)|,$$

and as $\tilde{u} \to u$ we obtain exact equality, where x and $\tilde{x}$ denote the first coordinates of u and $\tilde{u}$, respectively. Hence, we have

$$\frac{\partial u}{\partial x} = \arcsin(x)' = \frac{1}{\sqrt{1-x^2}}.$$

On the other hand, whith a fixed first coordinate $x \in [-1,1]$, the point u lays on a $(N-1)$-sphere with radius $\sqrt{1-x^2}$ which has the volume

$$R(x) = V(N-1) \cdot (\sqrt{1-x^2})^{N-1}.$$

Since the density of the distribution of U_1 has to be linear in both $\frac{\partial u}{\partial x}$ and $R(x)$, we obtain

$$\begin{aligned} f_1(x) &= s \cdot \arcsin(x)' \cdot (\sqrt{1-x^2})^{N-1} \\ &= B\left(\frac{1}{2}, \frac{N}{2}\right)^{-1} \cdot (\sqrt{1-x^2})^{N-2} \end{aligned}$$

for $x \in [-1,1]$. The factor s is the appropriate scaling factor assuring that the integral over f_1 is 1 and thus evaluates to

$$s = \left(\int_{-1}^{1} (\sqrt{1-x^2})^{N-2} dx\right)^{-1} = B\left(\frac{1}{2}, \frac{N}{2}\right)^{-1}.$$

Now it is straightforward to generate random numbers with density f_1. We have to find F_1 by integrating f_1 and finally get F_1^{-1} by inverting F_1. Thus, $U_1 = F_1^{-1}(Z)$ will have the desired properties, if Z is uniformly distributed on $[0,1]$. Performing the integration yields

$$F_1(x) = \int_{-1}^{x} f_1(y) dy = \frac{1}{2} \; + \; \operatorname{sign}(x) \cdot \frac{B_{x^2}\left(\frac{1}{2}, \frac{N}{2}\right)}{2 \cdot B\left(\frac{1}{2}, \frac{N}{2}\right)}.$$

The last step of inverting F_1 cannot be done in a closed form. So one has to employ an approximation algorithm such as Newton's method to perform this task numerically.

Once constructed a properly distributed U_1, the rest is done by recursion: The remaining task is to generate a random point distributed uniformly on the $(N-1)$-sphere with radius $\sqrt{1-x^2}$. The recursion will finally stop at $N = 0$, where it remains to randomly choose a point out of $\{-1,1\}$. For the reason of efficiency and accuracy, one can also treat the cases $N = 1$ and $N = 2$ seperately: For $N = 1$ one has to construct one coordinate of a point on the unit circle, while for $N = 2$ the formulas yield $f_1 \equiv \frac{1}{2}$ and $F_1(x) = \frac{1}{2}(x+1)$.

The time complexity of this algorithm is clearly linear, as solving the equation can be done in constant time. The main drawback of a practical implementation of this algorithm is the approximation that has to be done to solve the equation. Even a rapidly convergent approximation such as Newton's method restrains the performance considerably.

Note:

$$\begin{aligned} var(randball(d)) &= \frac{\sqrt{\pi}\Gamma(\frac{d+1}{2})}{2\Gamma(\frac{d+4}{2})B(\frac{1}{2},\frac{d+1}{2})} \\ &= \frac{\Gamma(1+\frac{d}{2})}{2\Gamma(2+\frac{d}{2})} \end{aligned}$$

4 Using normally distributed random vectors

This is the algorithm mentioned in Knuth ([3]). It makes use of a strong property of a normally distributed random vector. We briefly recall the definition.

Definition. A random variable has distribution $N(0,1)$ if it has the density function

$$f(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}x^2}.$$

A d-dimensional random vector X has distribution $N(0,I)$ if the components are independant and have distribution $N(0,1)$ each. In this case, the density of X is given by

$$f(x) = \frac{1}{(\sqrt{2\pi})^d} e^{-\frac{1}{2}<x,x>}.$$

In fact, the latter condition is also sufficient for X having independant and normally distributed components. This follows from the uniqueness of the Fourier transform and the particular form of the Fourier transform of the normal distribution (see [1]).

Theorem. *Let X be a d-dimensional random vector with distribution $N(0,I)$ and $U \in \mathbb{R}^{d\times d}$ an orthogonal matrix (i.e. $UU^t = U^tU = I$). Then $Y = UX$ has distribution $N(0,I)$, too.*

Proof. For any measurable set $A \subset \mathbb{R}^d$ we have

$$\begin{aligned} P(Y \in A) &= P(X \in U^tA) \\ &= \int_{U^tA} \frac{1}{(\sqrt{2\pi})^d} e^{-\frac{1}{2}<x,x>} \\ &= \int_A \frac{1}{(\sqrt{2\pi})^d} e^{-\frac{1}{2}<Ux,Ux>} \\ &= \int_A \frac{1}{(\sqrt{2\pi})^d} e^{-\frac{1}{2}<x,x>} \end{aligned}$$

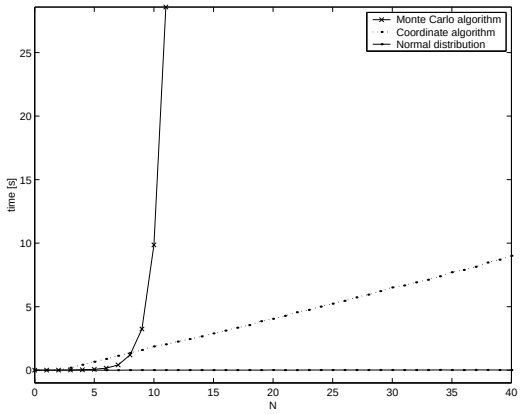

Figure 1: The performance of the three algorithms

by orthogonalty of U, hence the conclusion follows. □

As any rotation is in fact just a multiplication with an appropriate orthogonal matrix, we conclude from this theorem that normally distributed random vectors are invariant under rotation. Thus, generating $X \in \mathbb{R}^{N+1}$ with distribution $N(0, I)$ and then projecting it onto the sphere S_N produces random vectors $U = \|X\|^{-1} \cdot X$ that are uniformly distributed on the sphere.

For generating the random vector X, we can make use of the Box-Muller method (see [2]). Thus, the time time comlexity of the algorithm is clearly linear.

5 Comparison of the algorithms

As the latter of our algorithms neither rejects points nor involves slow approximation methods, we expect it to be the most efficient one. This is perfectly affirmed by Fig. 1. The figure displayes the time (in seconds) it took to generate 1000 random points on the N-sphere with varying N. As the experiments were carried out in Matlab, the data is a little biased: While the normal distribution algorithm profits from the very fast built in generator for normally distributed random numbers, the code for the incomplete beta function is very slow. Nontheless, the relations are correct. The time used by the Monte Carlo algorithm drastically increases at about $N = 10$, the algorithm is not usable for larger N. The coordinate algorithm takes linear time. The same does the normal distribution algorithm, but with a much smaller ascent.

6 Conclusions

For solving the problem of generating uniformly distributed random points on a high dimensional unit sphere in practice, the last algorithm is obviously the most efficient and therefore to be preferred. However, if a specific non-uniform distribution is needed, a modification of the coordinate alogrithm may be appropriate. Even a modification of the Monte Carlo algorithm can be suitable, even though it will probably result in an inefficient algorithm for high dimensions.

References

[1] H. Bauer. *Wahrscheinlichkeitstheorie*. deGruyter, 4th edition, 1991.

[2] G. Box and M. Muller. A note on the generation of random normal deviates. *Annals of Mathematical Statistics*, 29:610–611, 1958.

[3] D. E. Knuth. *The Art of Computer Programming, vol. 2: Seminumerical Algorithms*. Addison-Wesley, 1969.

[4] I. Rechenberg. *Evolutionsstrategie '94*. frommann-holzboog, Stuttgart, 1994.

[5] D. Rusin. Topics on sphere distributions. http://www.math.niu.edu/ rusin/known-math/95/sphere.faq.

[6] D. Seaman. Topics on sphere distributions. http://www.math.niu.edu/ rusin/known-math/96/sph.rand.

[7] A. Zell. *Simulation neuronaler Netze*. Addison-Wesley, Bonn, 1994.