Generating random vectors uniformly distributed inside and on the surface of different regions

R.Y. RUBINSTEIN *
IBM Thomas J. Watson Research Center, Yorktown Heights, NY 10598, U.S.A.

Received January 1981
Revised July 1981

An acceptance–rejection algorithm for generating random vectors uniformly distributed over (inside or on the surface of) a complex region inserted in a minimal multidimensional rectangle is considered. For regions having simple forms (simplex, hypersphere, hyperellipsoid) several algorithms are presented as well.

1. Introduction

A similator is often faced with the problem of generating random vectors uniformly distributed over (inside or on the surface of) different regions. No paper is known to the author at the present time which deals with this important problem despite the fact that statistical results connected with it are known well. In this paper we shall make an attempt to close partly this gap and present several algorithms based on the well-known Von Neumann's acceptance–rejection method and on some statistical results.

The objective of this part is two-fold. First in Section 2 we apply Von Neumann's acceptance–rejection method [3] for generating random variates uniformly distributed over complex regions. We also discuss the difficulties of obtaining high efficiency using Von Neumann's method and give some practical recommendations for achieving it. Secondly, in Section 3 we consider some particular cases assuming that the regions are rather nice (for example, multidimensional rectangle, hypersphere, and ellipsoid) and present some algorithms for generating uniform random variates over these regions.

2. The general case

Let us consider the problems of generating a random vector uniformly distributed over the complex region G (Fig. 1). The simplest way to do it is to insert the region G in a larger and nicer one, say Ω, to generate a random vector uniformly distributed in Ω and to accept or reject the generated vector depending on whether it hits or misses the region G. The acceptance–rejection algorithm is straightforward and can be written as:

Algorithm 1

1. Generate a random vector X uniformly distributed in Ω.
2. If $X \in G$ accept X as a variable uniformly distributed in G.
3. Go to Step 1.

We can easily see that the efficiency of this

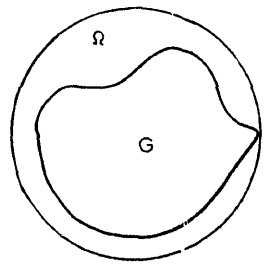

Fig. 1.

* Current address: Faculty of Industrial and Management Engineering, Technion, Haifa, Israel.

I wish to express my indebtedness to Professor Vidal Cohen of the University of Paris for many valuable suggestions, and his corrections to the earlier draft of the manuscript.

North-Holland Publishing Company
European Journal of Operational Research 10 (1982) 205–209

method is

$$C=\frac{\text{volume } G}{\text{volume } \Omega}, \quad 0<C<1$$

where C depends on the choice of the form of Ω. For this algorithm to be of practical interest the following criteria must be satisfied:

(1) It should be easy to generate a vector uniformly distributed in Ω;

(2) The efficiency of the procedure; C should be large.

For the first criterion to be met we can choose Ω to be a nice, usually convex region, say a multidimensional rectangle, a hypersphere, etc. For the second criterion to be met the volume of Ω has to be minimal, and such that $G \subset \Omega$. If G is complex both criteria are difficult to meet and sometimes are even in contradiction. In this paper we shall consider only one particular case, where Ω is a multidimensional rectangle and G is a region given by a system of the following non-linear equations $G=\{x: g_i(x)\leqslant 0, i=1,\ldots,m\}$, where $g_i(x)$ are known functions. Our problem can be formulated as the following. For a given $G=\{x: g_i(x)\leqslant 0, i=1,\ldots,m\}$ find a multidimensional rectangle Ω, which contains G and has a minimal volume.

For simplicity we shall consider the two dimensional case (see Fig. 2), i.e. where Ω is a rectangle. The problem is then equivalent to the problem of finding the extreme points a, b, c, d (see Fig. 2). Having these points at our disposal we can plot a minimal rectangle Ω containing G. To find these four points we can solve the following four nonlinear programming problems:

(1) $\max x_1$, s.t. $g_i(x)\leqslant 0, i=1,\ldots,m$;

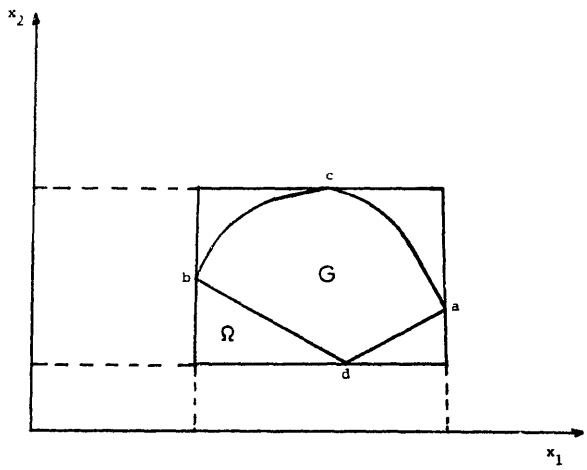

Fig. 2.

(2) $\min x_1$, s.t. $g_i(x)\leqslant 0, i=1,\ldots,m$;

(3) $\max x_2$, s.t. $g_i(x)\leqslant 0, i=1,\ldots,m$;

(4) $\min x_2$, s.t. $g_i(x)\leqslant 0, i=1,\ldots,m$.

If $g_i(x)\leqslant 0, i=1,\ldots,m$ are convex functions, we can apply the well-known methods of convex programming [1]. The extension to the n-dimensional case ($n>2$) is straightforward: we have to solve $2n$ similar problems. Generally the problem of inserting a convex region G into a larger one Ω which has the minimal volume and is also easy to generate in it is not easy to solve. The difficulties are even further aggravated if G is not convex. We hope to address these generalizations and other results in future papers. In the next section we shall consider some particular cases for uniform generation over G assuming that G itself has a simple shape.

3. Some particular cases

3.1. Generating random vectors over a simplex [4]

The density of a random vector uniformly distributed inside an n-dimensional simplex

$$S_n=\left\{(x_1,\ldots,x_n): x_i\geqslant 0, i=1,\ldots,n, \sum_{i=1}^{n} x_i\leqslant 1\right\} \tag{1}$$

is

$$f_X(x)=\begin{cases} n! & x=\{x_1,\ldots,x_n\}\in S_n, \\ 0, & \text{otherwise.}\end{cases} \tag{2}$$

This density is a particular case of Dirichlet density with parameters $v_1,\ldots,v_n, v_{n+1}$

$$f_X(x)=\frac{\Gamma(v_1+\cdots+v_{n+1})}{\Gamma(v_1)\cdots\Gamma(v_{n+1})}x_1^{v_1-1}\cdots x_n^{v_n-1} \times(1-x_1-\cdots-x_n)^{v_{(n+1)}+1} \tag{3}$$

and denoted $D(v_1,\ldots,v_n, v_{n+1})$.

Indeed for $v_1=\cdots=v_n=v_{n+1}=1$ (3) is equal (2). We shall denote the density (2) $D(1,\ldots,1, 1)$. It is also known [4] that if Y_i, $i=1,\ldots,n+1$ are independent r.v.'s distributed according to gamma distributions with parameters v_i and 1, i.e.

$$f_{Y_i}(y)=\begin{cases}\dfrac{y^{v_i-1}e^{-y}}{\Gamma(v_i)}, & y>0, v_i>0, \\ 0, & \text{otherwise,}\end{cases} \tag{4}$$

then the random vector

$$X=(X_1,\ldots,X_n)=\left(\frac{Y_1}{\sum_{i=1}^{n+1}Y_i},\ldots,\frac{Y_n}{\sum_{i=1}^{n+1}Y_i}\right) \tag{5}$$

is distributed $D(v_1,\ldots,v_n,\ v_{n+1})$. Taking into account that in the particular case where $v_i=1$, $i=1,\ldots,n+1$, $f_{Y_i}(y)$ is distributed exp(1) the algorithm for generating r.v.'s from $D(1,\ldots,1,\ 1)$ can be written as

Algorithm 2

1. Generate $(n+1)$ random variables from exp(1).
2. Apply formula (5) and deliver X as a vector distributed from $D(1,\ldots,1,\ 1)$.

It is readily seen that if we shall expand the n-dimensional vector in (5) to the following $(n+1)$-dimensional vector

$$X=(X_1,\ldots,X_{n+1})=\left(\frac{Y_1}{\sum_{i=1}^{n+1}Y_i},\ldots,\frac{Y_{n+1}}{\sum_{i=1}^{n+1}Y_i}\right), \tag{6}$$

we shall obtain a uniform distribution *on* the surface of the $(n+1)$-dimensional simplex

$$S_{n+1}=\left\{(x_1,\ldots,x_{n+1}):\ x_i\geqslant 0,\ i=1,\ldots,n+1,\ \sum_{i=1}^{n+1}x_i=1\right\}$$

instead of the uniform distribution *inside* the n-dimensional simplex considered in (5).

An alternative procedure for generating random vectors uniformly distributed *inside* an n-dimensional simplex is based on the following result [4]. Let $Y_{(1)},\ldots,Y_{(n)}$ be the order statistics from any continuous c.d.f. $F_Y(y)$. Then the random vector $X=(X_1,\ldots,X_n)$ with

$$\begin{aligned}
X_1&=F_Y(Y_{(1)}),\\
X_2&=F_Y(Y_{(2)})-F_Y(Y_{(1)}),\\
&\vdots\\
X_n&=F_Y(Y_{(n)})-F_Y(Y_{(n-1)}),
\end{aligned} \tag{7}$$

is distributed $D(1,\ldots,1,\ 1)$. Taking into account that the random variable $F_Y(Y)$ is distributed $U(0,\ 1)$ we can rewrite (7) as

$$\begin{aligned}
X_1&=U_{(1)},\\
X_2&=U_{(2)}-U_{(1)},\\
&\vdots\\
X_n&=U_{(n)}-U_{(n-1)}
\end{aligned} \tag{8}$$

The following algorithm is an alternative possibility for generating from $D(1,\ldots,1,\ 1)$.

Algorithm 3

1. Generate n independent r.v.'s $U_1,\ldots,U_n$ from $U(0,\ 1)$.
2. Arrange them to be order statistics $U_{(1)},\ldots,U_{(n)}$.
3. Apply formula (8) and deliver X as a vector distributed $D(1,\ldots,1,\ 1)$.

To obtain a vector uniformly distributed *on* the surface of $(n+1)$-dimensional simplex we expand as in (6) the n-dimensional (8) to the following $(n+1)$-dimensional

$$\begin{aligned}
X_1&=U_{(1)},\\
X_2&=U_{(2)}-U_{(1)},\\
&\vdots\\
X_n&=U_{(n)}-U_{(n-1)},\\
X_{n+1}&=1-U_{(n)}.
\end{aligned} \tag{9}$$

3.2. Generating random vectors uniformly distributed over an n-dimensional unit hypersphere

The density function of a random vector uniformly distributed on the surface of the n-dimensional sphere of radius r is

$$f_Y(y)=\begin{cases}\dfrac{\Gamma(n/2)}{2\pi^{n/2}r^{n-1}}, & y=\left\{(y_1,\ldots,y_n);\ \sum_{i=1}^{n}y_i^2=r^2\right\},\\ 0, & \text{otherwise.}\end{cases} \tag{10}$$

Formula (10) can be easily obtained if we take into account that the surface of the n-dimensional sphere of radius r is

$$\frac{2\pi^{n/2}r^{n-1}}{\Gamma(n/2)}.$$

The first algorithm for generating a vector uniformly distributed over an n-dimensional unit sphere is based on Von Neumann's acceptance–rejection method [3] and represents a particular case of Algorithm 1. In this case (see Fig. 1) Ω is an n-dimensional rectangular $\{-1<y_i<1\}_{i=1}^n$ and G is a unit hypersphere with the center at the origin.

We shall first consider an algorithm for generating a uniform vector *inside* the surface of the unit hypersphere which can be written as follows:

Algorithm 4

1. Generate $U_1,\dots,U_n$ from $U(0,1)$.
2. $X_1 \leftarrow 1-2U_1,\dots,X_n \leftarrow 1-2U_n$, and $Y^2 \leftarrow \Sigma_{i=1}^n X_i^2$.
3. If $Y^2<1$, accept $Y=(X_1,\dots,X_n)$ as the desired vector.
4. Go to Step 1.

The efficiency of the method is equal to the ratio

$$C=\frac{\text{volume of the hypersphere}}{\text{volume of the hypercube}}=\frac{\pi^{n/2}}{n2^{n-1}\Gamma(n/2)}.$$

For even n ($n=2m$),

$$C=\frac{\pi^m}{m!2^m}=\frac{1}{m!}\left(\frac{\pi}{2}\right)^m 2^{-m} \quad \text{and} \quad \lim_{m\to\infty} C=0.$$

That is, for n large enough, the acceptance–rejection method is inefficient.

To generate a random vector uniformly distributed *on* the surface of the n-dimensional unit sphere, we have to rewrite only Step 3 in the Algorithm 4 as follows:

3. If $Y^2<1$, accept $Z=(Z_1,\dots,Z_n)$ where $Z_i=X_i/Y$, $i=1,\dots,n$.

Another algorithm for generating a vector uniformly distributed *on* the surface of a unit hypersphere is based on the following:

Theorem. *Let $X_1,\dots,X_n$ be i.i.d. r.v.'s distributed $N(0,1)$, then the vector*

$$Y=(Y_1,\dots,Y_n)=\left(\frac{X_1}{Z},\dots,\frac{X_n}{Z}\right) \tag{11}$$

where $Z=(\Sigma_{i=1}^n X_i^2)^{1/2}$ is distributed uniformly on the surface of the unit hypersphere $\Sigma_{i=1}^n y_i^2=1$.

Proof. Let us make the following transformation

$$\begin{cases} z=v, \\ x_i=vy_i,\ i=1,\dots,n. \end{cases} \tag{12}$$

The Jacobian of this transformation is $J=z^{n+1}$ and we have

$$f_{Y_1,\dots,Y_n,V}(y_1,\dots,y_n,v)$$
$$=v^n\frac{1}{(2\pi)^{n/2}}\exp\left\{-\sum_{i=1}^{-n}\frac{(y_iv)^2}{2}\right\}$$
$$=v^n\frac{1}{(2\pi)^{n/2}}\exp\left\{-\frac{-v^2}{2}\sum_{i=1}^{n}y_i^2\right\}. \tag{13}$$

Since

$$\sum_{i=1}^n y_i^2=\sum_{i=1}^n\left(\frac{x_i}{z}\right)^2=\frac{1}{z^2}\sum_{i=1}^n x_i^2=1,$$

we obtain

$$f_{Y_1,\dots,Y_n,V}(y_1,\dots,y_n,v)=v^n\frac{1}{(2\pi)^{n/2}}\,e^{-v^2/2}. \tag{14}$$

It follows from (13) and (14) that $(Y_1,\dots,Y_n)$ and V are independent and that the vector $(Y_1,\dots,Y_n)$ is uniformly distributed on the surface of the unit sphere $\Sigma_{i=1}^n y_i^2=1$.

Algorithm 5

1. Generate n random variables X_i, $i=1,\dots,n$ from N(0, 1).
2. Compute $Z=(\Sigma_{i=1}^n X_i^2)^{1/2}$.
3. Apply formula (11) and deliver Y as a desired vector.

An alternative procedure is as follows: Selecting a point on a 1-sphere ($x_1=1$ or $x_1=-1$) determine from this a random point on a 2-sphere and so on until the required dimensionality is reached. Given a point $(x_1, x_2,\dots,x_n)$ on an n-sphere, the point on the $(n+1)$-sphere is

$$\left(rx_1, rx_2,\dots,rx_n, s\sqrt{1-r^2}\right)$$

where s is a random sign and r has the p.d.f.

$$f_r(r)=\begin{cases} k\dfrac{r^{n-1}}{\sqrt{1-r^2}}, \\ \qquad 0\le r\le 1,\ k=\left(\displaystyle\int_0^1\frac{r^{n-1}}{\sqrt{1-r^2}}\,dr\right)^{-1}, \\ 0, \qquad \text{otherwise.} \end{cases}$$

3.3. Generating random vectors uniformly distributed over the hyperellipsoid

The equation of the hyperellipsoid with the center at origin can be represented as

$$X^{t}\Sigma X = r^2 \tag{15}$$

where Σ is a positive defined and symmetric $(n \times n)$ matrix, and t is the transpose operator. In the particular case where $\Sigma = I$ we obtain the hypersphere with radius r. Inasmuch as Σ is positively defined and symmetric, there exists a unique lower triangular matrix

$$C = \begin{vmatrix} c_{11} & 0 & \dots & 0 \\ c_{21} & c_{22} & \dots & 0 \\ \vdots & \vdots & & \vdots \\ c_{n1} & c_{n2} & \dots & c_{nn} \end{vmatrix} \tag{16}$$

such that $\Sigma = CC^{t}$. It can be readily proven that if the vector X is uniformly distributed over (i.e. inside or on) the n-dimensional sphere with radius r, then the vector $Y = CX$ is uniformly distributed over the hyperellipsoid (15). The algorithm is as follows.

Algorithm 6

1. Generate $X = (X_1, \dots, X_n)$ uniformly distributed over the hypersphere with radius r.
2. Find the matrix C from $CC^{t} = \Sigma$.
3. Deliver $Y = CX$ as a desired vector.

Remark. The elements c_{ij} of the matrix C can be calculated from the following recursive formula [2]:

$$c_{ij} = \frac{\sigma_{ij} - \sum_{k=1}^{j-1} c_{ik}c_{jk}}{\left(\sigma_{jj} - \sum_{k=1}^{j-1} c_{jk}^2\right)^{1/2}},$$

where $\sum_{k=1}^{0} c_{ik}c_{jk} = 0$, $1 \leqslant j \leqslant i \leqslant n$, and σ_{ij} are elements of Σ.

4. Summary

We propose several algorithms for generating random vectors uniformly distributed inside and on the surface of different regions. We had no intention of comparing these algorithms because such a problem is rather difficult.

However, a little thought will indicate that:

(a) Algorithm 2 is more efficient than Algorithm 3 since the latter requires the arrangement of $U_1, \dots, U_m$ to be order statistics which is generally more time-consuming than generating $(n+1)$ exponentional variates and subsequently applying formula (6) in Algorithm 2,

(b) Algorithm 4 is simple; however, it can be recommended only for small n $(n \leqslant 5)$. For $n > 5$ it is practically inefficient and has to be replaced by Algorithm 5.

References

[1] M. Avriel *Nonlinear Programming* (Prentice-Hall, Englewood Cliffs, NJ, 1976).

[2] R.Y. Rubinstein, *Simulation and Monte Carlo Methods* (Wiley, New York, 1981).

[3] J. Von Neumann, Various techniques used in connection with random digits, *Collected Works* (Pergamon Press, New York, 1963) 768–770.

[4] S.S. Wilks, *Mathematical Statistics* (Wiley, New York, 1962).