A fast algorithm for generating a uniform distribution inside a high-dimensional polytope

André van Hameren* and Ronald Kleiss†
University of Nijmegen, Nijmegen, the Netherlands

November 13, 2018

Abstract

We describe a uniformly fast algorithm for generating points \( \vec{x} \) uniformly in a hypercube with the restriction that the difference between each pair of coordinates is bounded. We discuss the quality of the algorithm in the sense of its usage of pseudo-random source numbers, and present an interesting result on the correlation between the coordinates.

*andrevh@sci.kun.nl
†kleiss@sci.kun.nl
1 Introduction

In this paper we shall discuss the problem of generating sets of points $\bar{x} = (x_1, x_2, \ldots, x_m)$ inside an $m$-dimensional hypercube with an additional restriction. The points $\bar{x}$ are required to satisfy the conditions

$$|x_k| < 1, \quad |x_k - x_l| < 1 \text{ for all } k, l.$$  

(1)

These conditions define a $m$-dimensional convex polytope $P$. The reason for tackling this problem is the following. In a recently developed Monte Carlo algorithm, SARGE \cite{1}, we address the problem of generating configurations of four-momenta $p_i^\mu, i = 1, 2, \ldots, n$ of $n$ massless partons at high energy, with a distribution that has, as much as possible, the form of a so-called QCD antenna:

$$\frac{1}{s_{12}s_{23}s_{34}\cdots s_{n-1,n}s_{n1}}, \quad s_{kl} = (p_k + p_l)^2,$$

where $s_{kl}$ is the invariant mass squared of partons $k$ and $l$, with the additional requirement that the total invariant mass squared of all the partons is fixed to $s$, and every $s_{kl}$ (also those not occurring explicitly in the antenna) exceeds some lower bound $s_0$: in this way the singularities of the QCD matrix elements are avoided. The SARGE algorithm has a structure that is, in part, similar to the RAMBO algorithm \cite{2}, where generated momenta are scaled so as to attain the correct overall invariant mass. Obviously, in SARGE this is more problematic because of the $s_0$ cut, but one should like to implement this cut as far as possible. Note that out of the $n(n-1)/2$ different $s_{kl}$, $n$ occur in the antenna, and each of these must of course be bounded by $s_0$ from below and some $s_M < s$ from above. The scale-invariant ratios of two of these masses are therefore bounded by

$$\frac{s_0}{s_M} \leq \frac{s_{ij}}{s_{kl}} \leq \frac{s_M}{s_0},$$  

(2)

The structure of the SARGE algorithm is such \cite{1} that there are $m = 2n - 4$ of these ratios to be generated. By going over to variables

$$x_{(\cdot \cdot \cdot)} = \log(s_{ij}/s_{kl})/\log(s_M/s_0),$$

and inspecting all ratios that can be formed from the chosen $m$ ones, we arrive at the condition of Eq.(1). Note that, inside SARGE, a lot of internal rejection is going on, and events satisfying Eq.(1) may still be discarded: however, if Eq.(1) is not satisfied, the event is certainly discarded, and it therefore pays to include this condition from the start.
2 The algorithm

The most straightforward way of implementing is of course the following: generate \( x_k, k = 1, \ldots, m \) by \( x_k \leftarrow 2\rho - 1 \), and reject if the conditions are not met. Here and in the following, each occurrence of \( \rho \) stands for a call to a source of iid uniform pseudo-random numbers between in \([0, 1)\). The drawback of this approach is that the efficiency, i.e. the probability of success per try, is given by \( 2^{-m} V_m(P) \) (where \( V_m(P) \) is the volume of the polytope \( P \)) and becomes very small for large \( m \), as we shall see.

To compute the volume \( V_m(P) \) we first realize that the condition \(|x_k - x_l| < 1\) is only relevant when \( x_k \) and \( x_l \) have opposite sign. Therefore, we can divide the \( x \) variables in \( m - k \) positive and \( k \) negative ones, so that

\[
V_{m,k}(P) = \int_0^1 dy_1 dy_2 \cdots dy_k dx_{k+1} dx_{k+2} \cdots dx_m \theta \left( 1 - \max_i x_i - \max_j y_j \right),
\]

\[
V_m(P) = \sum_{k=0}^m \frac{m!}{k!(m-k)!} V_k(P), \tag{3}
\]

where we have written \( y_k = -x_k \). By symmetry we can always relabel the indices such that \( x_m = \max_i x_i \) and \( y_1 = \max_j y_j \). The integrals over the other \( x \)'s and \( y \)'s can then easily be done, and we find

\[
V_{m,k}(P) = k(m-k) \int_0^1 dy_1 y_1^{k-1} \int_0^{1-y_1} dx_m x_m^{m-k-1}
\]

\[
= k \int_0^1 dy_1 y_1^{k-1} (1 - y_1)^{m-k} = \frac{k!(m-k)!}{m!}, \tag{4}
\]

and hence

\[
V_m(P) = m + 1. \tag{5}
\]

The efficiency of the straightforward algorithm is therefore equal to \((m + 1)/2^m\), which is less than 3% for \( n \) larger than 6.

We have given the above derivation explicitly since it allows us, by working backwards, to find a rejection-free algorithm with unit efficiency. The algorithm is as follows:

1. Choose a value for \( k \). Since each \( k \) is exactly equally probably we simply have

\[
k \leftarrow \lfloor (m + 1) \rho \rfloor.
\]
2. For $k = 0$ we can simply put

$$x_i \leftarrow \rho \ , \ i = 1, \ldots, m \ ,$$

while for $k = m$ we put

$$x_i \leftarrow -\rho \ , \ i = 1, \ldots, m .$$

3. For $0 < k < m$, $y_1$ has the unnormalized density $y_1^{k-1}(1 - y_1)^{m-k}$ between 0 and 1. An efficient algorithm to do this is Cheng’s rejection algorithm BA for beta random variates (cf. [3]) but the following also works:

$$v_1 \leftarrow -\log \left( \prod_{i=1}^{k} \rho \right) \ , \ v_2 \leftarrow -\log \left( \prod_{j=1}^{m-k+1} \rho \right) \ , \ y_1 \leftarrow \frac{v_1}{v_1 + v_2} .$$

The variable $x_m$ has unnormalized density $x_m^{m-k-1}$ between 0 and 1 in $y_1$, so that it is generated by

$$x_m \leftarrow (1 - y_1)\rho^{1/(m-k)} .$$

The other $x$’s are now trivial:

$$x_1 \leftarrow -y_1 \ , \ x_i \leftarrow x_1 \rho , \ i = 2, 3, \ldots, k \ ,$$

$$x_i \leftarrow x_m \rho , \ i = k + 1, k + 2, \ldots, m - 1 .$$

Finally, perform a random permutation of the whole set $(x_1, x_2, \ldots, x_m)$.

3 Computational complexity

The number usage $S$, that is, the expected number of calls to the random number source $\rho$ per event can be derived easily. In the first place, 1 number is used to get $k$ for every event. In a fraction $2/(m+1)$ of the cases, only $m$ calls are made. In the remaining cases, there are $k + (m - k + 1) = m + 1$ calls to get $y_1$, and 1 call for all the other $x$ values. Finally, the simplest
permutation algorithm calls \( m - 1 \) times \[4\]. The expected number of calls is therefore

\[
S = 1 + \frac{2m}{m+1} + \frac{m-1}{m+1}(m+1 + (m-1) + (m-1)) = \frac{3m^2 - m + 2}{m+1} . \tag{6}
\]

For large \( m \) this comes to about \( 3m - 1 \) calls per event. Using a more sophisticated permutation algorithm would use at least 1 call, giving

\[
S = 1 + \frac{2m}{m+1} + \frac{m-1}{m+1}(m+1 + (m-1) + (1)) = 2m . \tag{7}
\]

We observed that Cheng’s rejection algorithm to obtain \( y_i \) uses about 2 calls per event. Denoting this number by \( C \) the expected number of calls becomes

\[
S = \frac{2m^2 + (C-1)m-C+3}{m+1} \sim 2m + C - 1 \tag{8}
\]

for the simple permutation algorithm, while the more sophisticated one would yield

\[
S = \frac{m^2 + (C+2)m-C+1}{m+1} \sim m + C + 2 . \tag{9}
\]

We see that in all these cases the algorithm is uniformly efficient in the sense that the needed number of calls is simply proportional to the problem’s complexity \( m \), as \( m \) becomes large. An ideal algorithm would of course still need \( m \) calls, while the straightforward rejection algorithm rather has \( S = m2^m/(m+1) \sim 2^m \) expected calls per event.

In the testing of algorithms such as this one, it is useful to study expectation values of, and correlations between, the various \( x_i \). Inserting either \( x_i \) or \( x_ix_j \) in the integral expression for \( V(P) \), we found after some algebra the following expectation values:

\[
E(x_i) = 0 \quad , \quad E(x_i^2) = \frac{m+3}{6(m+1)} \quad , \quad E(x_ix_j) = \frac{m+3}{12(m+1)} \quad (i \neq j) . \tag{10}
\]

so that the correlation coefficient between two different \( x \)’s is precisely \( 1/2 \) in all dimensions! This somewhat surprising fact allows for a simple but powerful check on the correctness of the algorithm’s implementation.

As an extra illustration of the efficiency, we present in the tables below the cpu-time (\( t_{cpu} \)) needed to generate 1000 points in an \( m \)-dimensional polytope, both with the algorithm presented in this paper (\texttt{OURALG}) and the rejection method (\texttt{REJECT}). In the latter, we just
1. put $x_i \leftarrow 2\rho - 1$ for $i = 1, \ldots, m$;

2. reject $\vec{x}$ if $|x_i - x_j| > 1$ for $i = 1, \ldots, m - 1$ and $j = i + 1, \ldots, m$.

The computations were done using a single 333-MHz UltraSPARC-IIi processor, and the random number generator used was RANLUX on level 3.

| $m$ | $t_{cpu}$ (sec) | OURALG | REJECT |
|-----|----------------|--------|--------|
| 2   | 0.03           | 0.01   |        |
| 3   | 0.03           | 0.02   |        |
| 4   | 0.03           | 0.04   |        |
| 5   | 0.04           | 0.08   |        |
| 6   | 0.05           | 0.17   |        |
| 7   | 0.06           | 0.32   |        |
| 8   | 0.07           | 0.67   |        |
| 9   | 0.08           | 1.33   |        |
| 10  | 0.09           | 2.76   |        |

For $m = 2$ and $m = 3$, the rejection method is quicker, but from $m = 4$ on, the cpu-time clearly grows linearly for the method presented in this paper, and exponentially for the rejection method.

4 Extension

Let us, finally, comment on one possible extension of this algorithm. Suppose that the points $\vec{x}$ are distributed on the polytope $P$, but with an additional (unnormalized) density given by

$$F(\vec{x}) = \prod_{i=1}^{m} \cos \left( \frac{\pi x_i}{2} \right) , \quad (11)$$

so that the density is suppressed near the edges. It is then still possible to compute $V_{m,k}(P)$ for this new density:

$$V_{k,m}(P) = k(m - k) \int_{0}^{1} \int_{0}^{1-y_1} dy_1 \cos \left( \frac{\pi y_1}{2} \right) \int_{0}^{1-y_1} dx_m \cos \left( \frac{\pi x_m}{2} \right)$$
\[
\left( \int_0^{y_1} dy \cos \left( \frac{\pi y}{2} \right) \right)^{k-1} \left( \int_0^{x_m} dx \cos \left( \frac{\pi x}{2} \right) \right)^{m-k-1}
\]
\[
= k(m-k) \left( \frac{2}{\pi} \right)^{m-1} \int_0^1 d\sin \left( \frac{\pi y_1}{2} \right) \left( \sin \left( \frac{\pi y_1}{2} \right) \right)^{k-1}
\]
\[
\int_0^1 d\sin \left( \frac{\pi x_m}{2} \right) \left( \sin \left( \frac{\pi x_m}{2} \right) \right)^{m-k-1}
\]
\[
= \frac{2^{m-1}k}{\pi^m} \int_0^1 ds \ s^{k/2-1}(1-s)^{(m-k)/2}
\]
\[
= \left( \frac{2}{\pi} \right)^m \frac{\Gamma(1+k/2)\Gamma(1+(m-k)/2)}{\Gamma(1+m/2)} , \quad (12)
\]
where we used \( s = \left( \sin \left( \frac{\pi m}{2} \right) \right)^2 \). Therefore, a uniformly efficient algorithm can be constructed in this case as well, along the following lines. Using the \( V_{k,m} \), the relative weights for each \( k \) can be determined. Then \( s \) is generated as a \( \beta \) distribution. The generation of the other \( x \)'s involves only manipulations with sine and arcsine functions. Note that, for large \( m \), the weighted volume of the polytope \( P \) is

\[
V(P) = \sum_{k=0}^{m} \left( \frac{2}{\pi} \right)^m \frac{(k)!}{(m/2)!} \frac{(m-k)!}{k!} \frac{m!}{(m-k)!} \approx m \sqrt{\frac{\pi}{8}} \left( \frac{8}{\pi^2} \right)^{m/2} , \quad (13)
\]
so that a straightforward rejection algorithm would have number usage

\[
S \sim \sqrt{\frac{8}{\pi}} \left( \frac{\pi^2}{2} \right)^{m/2} , \quad (14)
\]
and a correspondingly decreasing efficiency.

**References**

[1] P. Dragiotis, A. van Hameren and R. Kleiss, preprint [hep-ph/0004047].
[2] S.D. Ellis, R. Kleiss and W.J. Stirling, *A new Monte Carlo treatment of multiparticle phase space at high energy*, Comp. Phys. Comm. 40 (1986) 359.

[3] L. Devroye, *Non-Uniform Random Variate Generation*, (Springer, 1986).

[4] D.E. Knuth, *The Art of Computer Programming, Vol.2. 2d ed.*, (Princeton, 1991).