Chapter Eight
TABLE METHODS FOR CONTINUOUS RANDOM VARIATES

1. COMPOSITION VERSUS REJECTION.

We have illustrated how algorithms can be sped up if we are willing to compute certain constants beforehand. For example, when a discrete random variate is generated by the inversion method, it pays to compute and store the individual probabilities p_n beforehand. This information can speed up sequential search, or could be used in the method of guide tables. For continuous random variates, the same remains true. Because we know many ultra fast discrete random variate generation methods, but very few fast continuous random variate generation techniques, there is a more pressing need for acceleration in the continuous case. Globally speaking, discretizing the problem speeds generation.

We can for example cut up the graph of f into pieces, and use the composition method. Choosing a piece is a discrete random variate generation problem. Generating a continuous random variate for an individual piece is usually simple because of the shape of the piece which is selected by us. There are only a few drawbacks: first of all, we need to know the areas of the pieces. Typically, this is equivalent to knowing the distribution function. Very often, as with the normal density for example, the distribution function must be computed as the integral of the density, which in our model is an infinite time operation. In particular, the composition method can hardly be made automatic because of this. Secondly, we observe that there usually are several nonrectangular pieces, which are commonly handled via the rejection method. Rectangular pieces are of course most convenient since we can just return a properly translated and scaled uniform random variate. For this reason, the total area of the nonrectangular pieces should be kept as small as possible.

There is another approach which does not require integration of f. If we find a function $g \geq f$, and use rejection, then similar accelerations can be obtained if we cut the graph of g up into convenient pieces. But because g is picked by us, we do of course know the areas (weights) of the pieces, and we can choose g piecewise constant so that each component piece is for example

rectangular. One could object that for this method, we need to compute the ratio f/g rather often as part of the rejection algorithm. But this too can be avoided whenever a given piece lies completely under the graph of f. Thus, in the design of pieces, we should try to maximize the area of all the pieces entirely covered by the graph of f.

From this general description, it is seen that all boils down to decompositions of densities into small manageable pieces. Basically, such decompositions account for nearly all very fast methods available today: Marsaglia's rectangle-wedge-tail method for normal and exponential densities (Marsaglia, Maclaren and Bray, 1964; Marsaglia, Ananthanarayanan and Paul, 1976), the method of Ahrens and Kohrt (1981), the alias-rejection-mixture method (Kronmal and Peterson, 1980), and the ziggurat method (Marsaglia and Tsang, 1984). The acceleration can only work well if we have a finite decomposition. Thus, infinite tails must be cut off and dealt with separately. Also, from a didactical point of view, rectangular decompositions are by far the most important ones. We could add triangles, but this would detract from the main points. Since we do care about the generality of the results, it seems pointless to describe a particular normal generator for example. Instead, we will present algorithms which are applicable to large classes of densities. Our treatment differs from that found in the references cited above. But at the same time, all the ideas are borrowed from those same references.

In section 2, we will discuss strip methods, i.e. methods that are based upon the partition of f into parallel strips. Because the strips have unequal probabilities, the strip selection part of the algorithm is usually based upon the alias or alias-urn methods. Partitions into equal parts are convenient because then fast table methods can be used directly. This is further explored in section 3.

2. STRIP METHODS.

2.1. Definition.

The following will be our standing assumptions in this section: f is a bounded density on [0,1]; the interval [0,1] is divided into n equal parts (n is chosen by the user); g is a function constant on the n intervals, 0 outside [0,1], and at least equal to f everywhere. We set

$$g(x) = g_i \quad \left(\frac{i-1}{n} \le x < \frac{i}{n}\right) \ (1 \le i \le n) .$$

Define the strip probabilities

$$p_i = \frac{g_i}{\sum_{j=1}^{n} g_j} \quad (1 \le i \le n) .$$

Then, the following rejection algorithm is valid for generating a random variate with density f :

REPEAT

Generate a discrete random variate Z whose distribution is determined by $P(Z=i)=p_i \quad (1 \leq i \leq n)$.

Generate two iid uniform [0,1] random variate U, V.

$X \leftarrow \frac{Z-1+V}{n}$

UNTIL $Ug_Z \leq f(X)$

RETURN X

As n increases, the rejection rate should diminish since it is possible to find better and better dominating functions g. But regardless of how large n is picked, there is no avoiding the two uniform random variates and the computation of $f(X)$. Suppose now that each strip is cut into two parts by a horizontal line, and that the bottom part is completely tucked under the graph of f. For part i, the horizontal line has height h_i. We can set up a table of $2n$ probabilities: $p_1, \ldots, p_n$ correspond to the bottom portions, and $p_{n+1}, \ldots, p_{2n}$ to the top portions. Then, random variate generation can proceed as follows:

REPEAT

Generate a discrete random variate Z whose distribution is determined by $P(Z=i)=p_i \quad (1 \leq i \leq 2n)$.

Generate a uniform [0,1] random variate V.

$X \leftarrow \frac{Z-1+V}{n}$

IF $Z \leq n$

THEN RETURN X

ELSE

Generate a uniform [0,1] random variate U.

IF $h_{Z-n}+U(g_{Z-n}-h_{Z-n}) \leq f(X-1)$ THEN RETURN $X-1$

UNTIL False

When the bottom probabilities are dominant, we can get away with generating just one discrete random variate Z and one uniform [0,1] random variate V most of the time. The performance of the algorithm is summarized in Theorem 2.1:

Theorem 2.1.

For the rejection method based upon n split strips of equal width, we have:

1. The expected number of iterations is $\frac{1}{n}\sum_{i=1}^{n} g_i$. This is also equal to the expected number of discrete random variates Z per returned random variate X.
2. The expected number of computations of f is $\frac{1}{n}\sum_{i=1}^{n}(g_i - h_i)$.
3. The expected number of uniform [0,1] random variates is $\frac{1}{n}\sum_{i=1}^{n} g_i + \frac{1}{n}\sum_{i=1}^{n}(g_i - h_i)$.

Proof of Theorem 2.1.

The proof uses standard properties of rejection algorithms, together with Wald's equation. ■

The algorithm requires tables for g_i, h_i, $1 \le i \le n$, and p_i, $1 \le i \le 2n$. Some of the $4n$ numbers stored away contain redundant information. Indeed, the p_i's can be computed from the g_i's and h_i's. We store redundant information to increase the speed of the algorithm. There may be additional storage requirements depending upon the discrete random variate generation method: see for example what is needed for the method of guide tables, and the alias and alias-urn methods which are recommended for this application. Recall that the expected time of these generators does not depend upon n.

Thus, we are left only with the computation of the g_i's and h_i's. Consider first the best possible constants:

$$g_i = \sup_{\frac{i-1}{n} \le x < \frac{i}{n}} f(x);$$

$$h_i = \inf_{\frac{i-1}{n} \le x < \frac{i}{n}} f(x).$$

Normally, we cannot hope to compute these values in a finite amount of time. For specially restricted densities f, it is possible however to do so quite easily. Regardless of whether we can actually compute them or not, we have the following important observation:

Theorem 2.2.

Assume that f is a Riemann integrable density on [0,1]. Then, if g_i , h_i are defined by:

$$g_i = \sup_{\frac{i-1}{n} \le x < \frac{i}{n}} f(x) ;$$

$$h_i = \inf_{\frac{i-1}{n} \le x < \frac{i}{n}} f(x) ,$$

we have:

1. $\lim_{n \to \infty} \frac{1}{n} \sum_{i=1}^{n} g_i = 1;$
2. $\lim_{n \to \infty} \frac{1}{n} \sum_{i=1}^{n} (g_i - h_i) = 0 .$

Proof of Theorem 2.2.

It suffices to prove the second statement, in view of the fact that

$$\frac{1}{n} \sum_{i=1}^{n} g_i \le 1 + \frac{1}{n} \sum_{i=1}^{n} (g_i - h_i) .$$

But the second statement is a direct consequence of the definition of Riemann integrability. ■

Thus, for sufficiently well-behaved densities, if we have optimal bounds g_i, h_i at our disposal, the algorithm becomes very efficient when n grows large.

2.2. Example 1: monotone densities on [0,1].

When f is monotone on [0,1], we can set

$$g_i = f\left(\frac{i-1}{n}\right) ; h_i = f\left(\frac{i}{n}\right) .$$

We also have

$$\frac{1}{n} \sum_{i=1}^{n} (g_i - h_i) = \frac{1}{n} \sum_{i=1}^{n} \left(f\left(\frac{i-1}{n}\right) - f\left(\frac{i}{n}\right)\right)$$

$$= \frac{1}{n}(f(0) - f(1)) \le \frac{f(0)}{n} .$$

The performance of the algorithm can be summarized quite simply:

1. The expected number of iterations is $\leq 1+\frac{f(0)}{n}$. This is also equal to the expected number of discrete random variates Z per returned random variate X.
2. The expected number of computations of f is $\leq \frac{f(0)}{n}$.
3. The expected number of uniform [0,1] random variates is $\leq 1+\frac{2f(0)}{n}$.

We also note that to set up the tables g_i, h_i, it suffices to evaluate f at the $n+1$ mesh points. Furthermore, the extremes of f are reached at the endpoints of the intervals, so that the constants are in this case best possible. The only way to improve the performance of the algorithm would be by considering unequal interval sizes. It should be clear that the interval sizes should become smaller as we approach the origin. The unequal intervals need to be picked with care if real savings are needed. For a fair comparison, we will use n intervals with breakpoints

$$0=x_0<x_1<x_2<\cdots<x_n=1\ ,$$

where

$$x_{i+1}-x_i=\delta b^i \qquad (0\leq i\leq n-1)\ ,$$

$$\delta=\frac{b-1}{b^n-1}\ ,$$

and $b>1$ is a design constant. The algorithm is only slightly different now because an additional array of x_i's is stored.

Theorem 2.3.

Assume that f is a monotone density on [0,1]. Then for the rejection-based strip method shown above,

A. The expected number of iterations does not exceed

$$b+f(0)\frac{b-1}{b^n-1}.$$

B. If $b=1+\frac{1}{n}\log(1+f(0)+f(0)\log(f(0)))$, then the upper bound is of the form

$$1+\frac{1}{n}(\log(1+f(0)+f(0)\log(f(0))))(1+\frac{1}{1+\log(f(0))}+o(1))$$

as $n\to\infty$. (Note: when $f(0)$ is large, we have approximately $1+\frac{\log(f(0))}{n}$.)

Proof of Theorem 2.3.

The expected number of iterations is

$$\sum_{i=0}^{n-1} f(x_i)(x_{i+1}-x_i)$$

$$=\sum_{i=0}^{n-1}\delta b^i f(x_i)$$

$$\le \delta f(0)+\sum_{i=1}^{n-1} b\int_{x_{i-1}}^{x_i} f(y)\,dy$$

$$\le b+f(0)\frac{b-1}{b^n-1}.$$

When $b=1+\frac{c}{n}$ for some constant $c>0$, then it is easy to see that the upper bound is

$$1+\frac{1}{n}(c+f(0)\frac{c}{e^c-1+o(1)}).$$

Replace c by $\log(1+f(0)+f(0)\log(f(0)))$. ■

What we retain from Theorem 2.3 is that with some careful design, we can do much better than in the equi-spaced interval case. Roughly speaking, we have reduced the expected number of iterations for monotone densities on [0,1] from $1+\frac{f(0)}{n}$ to $1+\frac{\log(f(0))}{n}$. Several details of the last algorithm are dealt with in

the exercises.

2.3. Other examples.

In the absence of information about monotonicity or unimodality, it is virtually impossible to compute the best possible constants g_i and h_i for the rejection-based table method. Other pieces of information can aid in the derivation of slightly sub-optimal constants. For example, when $f \in Lip_1(C)$, then

$$g_i = \frac{C}{2n} + \frac{f\left(\frac{i-1}{n}\right) + f\left(\frac{i}{n}\right)}{2},$$

$$h_i = -\frac{C}{2n} + \frac{f\left(\frac{i-1}{n}\right) + f\left(\frac{i}{n}\right)}{2},$$

will do. These numbers can again be computed from the values of f at the $n+1$ mesh points. We can work out the details of Theorem 2.1:

Theorem 2.4.

For the rejection method based upon n split strips of equal width, used on a $Lip_1(C)$ density f on [0,1], we have:

1. The expected number of iterations is

$$\frac{C}{2n} + \frac{f(0) + 2f\left(\frac{1}{n}\right) + \cdots + 2f\left(\frac{n-1}{n}\right) + f(1)}{2} \leq 1 + \frac{C}{n}.$$

   This is also equal to the expected number of discrete random variates Z per returned random variate X.

2. The expected number of computations of f is $\leq \frac{C}{n}$.

3. The expected number of uniform [0,1] random variates is $\leq 1 + \frac{2C}{n}$.

Proof of Theorem 2.4.

The first expression follows directly after resubstitution of the values of g_i and h_i into Theorem 2.1. The upper bound of parts 1 and 2 are obtained by noting that $g_i - h_i = \frac{C}{n}$ for all i. Finally, part 3 is obtained by summing the bounds obtained in parts 1 and 2. ■

Once again, we can control the performance characteristics of the algorithm by our choice of n. The characteristics can be improved slightly if we make use of the fact that for Lipschitz densities known at mesh points, the obvious piecewise linear dominating curve has slightly smaller integral than the piecewise constant dominating curve suggested here. It should be noted that the switch to piecewise linear dominating curves is costly in terms of the number of uniform random variates needed, and in terms of the length of the program. It is much simpler to improve the performance by increasing n.

2.4. Exercises.

1. For the algorithm for monotone densities analyzed in Theorem 2.3, give a good upper bound for the expected number of computations of f, both in terms of general constants $b>1$ and for the constant actually suggested in Theorem 2.3.
2. When f is monotone and convex on [0,1], then the piecewise linear curve which touches the curve of f at the mesh points can be used as a dominating curve. If n equal intervals are used, show that the expected number of evaluations of f can be reduced by 50% over the corresponding piecewise constant case. Give the details of the algorithm. Compare the expected number of uniform [0,1] random variates for both cases.
3. Develop the details of the rejection-based strip method for Lipschitz densities which uses a piecewise linear dominating curve and n equi-spaced intervals. Compute good bounds for the expected number of iterations, the expected number of computations of f, and the expected number of uniform [0,1] random variates actually required.
4. **Adaptive methods.** Consider a bounded monotone density f on [0,1]. When $f(0)$ is known, we can generate a random variate by rejection from a uniform density on [0,1]. This corresponds to the strip method with one interval. As random variates are generated, the dominating curve for the strip method can be adjusted by considering a staircase function with breakpoints at the X_i's. This calls for a dynamic data structure for adjusting the probabilities and sampling from a varying discrete distribution. Design such a structure, and prove that the expected time needed per adjustment is $O(1)$ as $n\to\infty$, and that the expected number of f evaluations is $o(1)$ as $n\to\infty$.
5. Let F be a continuous distribution function. For fixed but large n, compute $x_i=F^{-1}(\frac{i}{n})$, $0\le i\le n$. Select one of the x_i's ($0\le i<n$) with equal probability $1/n$, and define $X=x_i+U(x_{i+1}-x_i)$ where U is a uniform [0,1] random variate. The random variable X has distribution function G_n which is close to F. It has been suggested as a fast universal table method in a variety of papers; for similar approaches, see Barnard and Cawdery (1974) and Mitchell (1977). When $x_0=-\infty$ or $x_n=\infty$, define X in a sensible way on the interval in question.

[SET-UP]

Choose $b > 1$, and integer $n > 1$. Set $\delta \leftarrow \frac{b-1}{b^n-1}$. Set $x_0 \leftarrow 0$.

FOR $i := 1$ TO n DO

$x_i \leftarrow \delta \frac{b^i-1}{b-1}$

$g_i \leftarrow f(x_{i-1})$; $h_i \leftarrow f(x_i)$

$p_i \leftarrow h_i(x_i - x_{i-1})$

$p_{n+i} \leftarrow (g_i - h_i)(x_i - x_{i-1})$

Normalize the vector of p_i's.

[GENERATOR]

REPEAT

Generate a discrete random variate Z whose distribution is determined by $P(Z=i)=p_i \quad (1 \leq i \leq 2n)$.

Generate a uniform [0,1] random variate V.

$W \leftarrow (Z-1) \bmod n$

$X \leftarrow x_W + V(x_{W+1} - x_W))$

IF $Z \leq n$

THEN RETURN X

ELSE

Generate a uniform [0,1] random variate U.

IF $h_{Z-n} + U(g_{Z-n} - h_{Z-n}) \leq f(X)$ THEN RETURN X

UNTIL False

A. Prove that in all cases, sup $|F-G_n| \to 0$ as $n \to \infty$.

B. Prove that when F has a density f, then $\int |f-g_n| \to 0$ as $n \to \infty$, where g_n is the density of G_n. This property holds true without exception.

C. Determine an upper bound on the L_1 error of part B in terms of f' and n whenever f is absolutely continuous with almost everywhere derivative f'.

3. GRID METHODS.

3.1. Introduction.

Some acceleration can be obtained over strip methods if we make sure that all the components boxes (usually rectangles) are of equal area. In that case, the standard (very fast) table methods can be used for generation. The cost can be prohibitive: the boxes must be fine so that they can capture the detail in the outline of the density f, and this forces us to store very many small boxes.

The versatility of the principle is illustrated here on a variety of problems, ranging from the problem of the generation of a uniformly distributed random vector in a compact set of R^d, to avoidance problems, and fast random variate generation.

3.2. Generating a point uniformly in a compact set.

Let us enclose the compact set A of R^d with a hyperrectangle H with sides $h_1, h_2, \ldots, h_d$. Divide each side up into N_i intervals of length $\frac{h_i}{N_i}, 1 \le i \le d$. There are three types of grid rectangles, the good rectangles (entirely contained in A), the bad rectangles (those partially overlapping with A), and the useless rectangles (those entirely outside A). Before we start generating, we need to set up an array of addresses of rectangles, which we shall call a directory. For the time being, we can think of an address of a rectangle as the coordinates of its leftmost vertex (in all directions). The directory (called D) is such that in positions 1 through k we have good rectangles, and in positions $k+1$ through $k+l$, we have bad rectangles. Useless rectangles are not represented in the array. The informal algorithm for generating a uniformly distributed point in A is as follows:

REPEAT

Generate an integer Z uniformly distributed in $1,2, \ldots, k+l$.

Generate X uniformly in rectangle $D[Z]$ ($D[Z]$ contains the address of rectangle Z).

Accept $\leftarrow[Z \leq k]$ (Accept is a boolean variable.)

IF NOT Accept THEN Accept $\leftarrow[X \in A]$.

UNTIL Accept

RETURN X

The expected number of iterations is equal to

$$\frac{\text{area}(C)}{\text{area}(A)}$$

where C is the union of the good and bad rectangles (if the useless rectangles are not discarded, then $C=H$). If the area of one rectangle is a, then $\text{area}(C)=a(k+l)$. For most bounded sets A, this can be made to go to 1 as the grid becomes finer. That this is not always the case follows from this simple example: let A be $[0,1]^d$ union all the rational vectors in $[1,2]^d$. Since the rationals are dense in the real line, any grid cover of A necessarily covers $[0,1]^d$ and $[1,2]^d$, so that the ratio of the areas is always at least 2. Fortunately, for all compact (i.e., closed and bounded) sets A, the given ratio of areas tends to one as the grid becomes finer (see Theorem 3.1).

The speed of the algorithm follows from the fact that when a good rectangle is chosen, no boundary checking needs to be done. Also, there are many more good rectangles than bad rectangles, so that the contribution to the expected time from boundary checking is small. Of course, we must in any case look up an entry in a directory. This is reminiscent of the urn or table look-up method and its modifications (such as the alias method (Walker, 1977) and the alias-urn method (Peterson and Kronmal, 1982)). Finer grids yield faster generators but require more space.

One of the measures of the efficiency of the algorithm is the expected number of iterations. We have to make sure that as the grid becomes finer, this expected number tends to one.

Theorem 3.1.

Let A be a compact set of nonzero area (Lebesgue measure), and let us consider a sequence of grids $G_1, G_2, \ldots$ which is such that as $n \to \infty$, the diameter of the prototype grid rectangle tends to 0. If C_n is the grid cover of A defined by G_n, then the ratio $\dfrac{\text{area}(C_n)}{\text{area}(A)}$ tends to 1 as $n \to \infty$.

Proof of Theorem 3.1.

Let H be an open rectangle covering A, and let B be the intersection of H with the complement of A. Then, B is open. Thus, for every $x \in B$, we know that the grid rectangle in G_n to which it belongs is entirely contained in B for all n large enough. Thus, by the Lebesgue dominated convergence theorem, the Lebesgue measure of the "useless" rectangles tends to the Lebesgue measure of B. But then, the Lebesgue measure of C_n must tend to the Lebesgue measure of A. ■

The directory itself can be constructed as follows: define a large enough array (of size $n = N_1 N_2 \cdots N_d$), initially unused, and keep two stack pointers, one for a top stack growing from position 1 down, and one for a bottom stack growing from the last position up. The two stacks are tied down at the ends of the array and grow towards each other. Travel from grid rectangle to grid rectangle, identify the type of rectangle, and push the address onto the top stack when it corresponds to a good rectangle, and onto the bottom stack when we have a bad rectangle. Useless rectangles are ignored. After this, the array is partially full, and we can move the bottom stack up to fill positions $k+1$ through $k+l$. If the number of useless rectangles is expected to be unreasonably large, then the stacks should first be implemented as linked lists and at the end copied to the directory of size $k+l$. In any case, the preprocessing step takes time equal to n, the cardinality of the grid.

It is important to obtain a good estimate of the size of the directory. We have

$$k+l \geq \frac{\text{area}(A)}{a} = \frac{\text{area}(A)}{\text{area}(H)}\, n \ .$$

We know from Theorem 3.1 and the fact that $\text{area}(C_n) = (k+l)a$, that

$$\lim_{n \to \infty} \frac{k+l}{n} = \frac{\text{area}(A)}{\text{area}(H)} \ ,$$

provided that as $n \to \infty$, we make sure that $\inf_i N_i \to \infty$ (this will insure that the diameter of the prototype rectangle tends to 0). Upper bounds on the size of the directory are harder to come by in general. Let us consider a few special cases in

the plane, to illustrate some points. If A is a convex set for example, then we can look at all N_1 columns and N_2 rows in the grid, and mark the extremal bad rectangles on either side, together with their immediate neighbors on the inside. Thus, in each row and column, we are putting at most 4 marks. Our claim is that unmarked rectangles are either useless or good. For if a bad rectangle is not marked, then it has at least two neighbors due north, south, east and west that are marked. By the convexity of A, it is physically impossible that this rectangle is not completely contained in A. Thus, the number of bad rectangles is at most $4(N_1+N_2)$. Therefore,

$$k+l \leq n\frac{\text{area}(A)}{\text{area}(H)} + 4(N_1+N_2) .$$

If A consists of a union of K convex sets, then a very crude bound for $k+l$ could be obtained by replacing 4 by $4K$ (just repeat the marking procedure for each convex set). We summarize:

Theorem 3.2.

The size of the directory is $k+l$, where

$$\frac{\text{area}(A)}{\text{area}(H)} \leq \frac{k+l}{n} = (1+o(1))\frac{\text{area}(A)}{\text{area}(H)} .$$

The asymptotic result is valid whenever the diameter of the grid rectangle tends to 0. For convex sets A on R^2, we also have the upper bound

$$\frac{k+l}{n} \leq \frac{\text{area}(A)}{\text{area}(H)} + 4\frac{N_1+N_2}{N_1N_2} .$$

We are left now with the choice of the N_i's. In the example of a convex set in the plane, the expected number of iterations is

$$\frac{(k+l)a}{\text{area}(A)} \leq 1+\frac{\text{area}(H)}{\text{area}(A)}\,\frac{4}{n}(N_1+N_2) .$$

The upper bound is minimal for $N_1=N_2=\sqrt{n}$ (assume for the sake of convenience that n is a perfect square). Thus, the expected number of iterations does not exceed

$$1+\frac{\text{area}(H)}{\text{area}(A)}\,\frac{8}{\sqrt{n}} .$$

This is of the form $1+\frac{\text{constant}}{\sqrt{n}}$ where n is the cardinality of the enclosing grid. By controlling n, we can now control the expected time taken by the algorithm. The algorithm is fast if we avoid the bad rectangles very often. It is easy to see that the expected number of inspections of bad rectangles before halting is the expected number of iterations times $\frac{l}{k+l}$, which equals to $\frac{l\ \text{area}(H)}{n\ \text{area}(A)} = o(1)$

since $\frac{l}{n} \to 0$ (as a consequence of Theorem 3.1). Thus, asymptotically, we spend a negligible fraction of time inspecting bad rectangles. In fact, using the special example of a convex set in the plane with $N_1 = N_2 = \sqrt{n}$, we see that the expected number of bad rectangle inspections is at most

$$\frac{\text{area}(H)}{\text{area}(A)} \frac{8}{\sqrt{n}} .$$

3.3. Avoidance problems.

In some simulations, usually with geometric implications, one is asked to generate points uniformly in a set A but not in $\cup A_i$ where the A_i's are given sets of R^d. For example, when one simulates the random parking process (cars of length one park at random in a street of length L but should avoid each other), it is important to generate points uniformly in $[0,L]$ minus the union of some intervals of the same length. Towards the end of one simulation run, when the street fills up, it is not feasible to keep generating new points until one falls in a good spot. Here a grid structure will be useful. In two dimensions, similar problems occur: for example, the circle avoidance problem is concerned with the generation of uniform points in a circle given that the point cannot belong to any of a given number of circles (usually, but not necessarily, having the same radius). For applications involving nonoverlapping circles, see Alder and Wainwright (1962), Diggle, Besag and Gleaves (1976), Talbot and Willis (1980), Kelly and Ripley (1976) and Ripley (1977, 1979). Ripley (1979) employs the rejection method for sampling, and Lotwick (1982) triangulates the space in such a way that each triangle has one of the data points as a vertex. The triangulation is designed to make sampling easy, and to improve the rejection constant. Lotwick also investigates the performance of the ordinary rejection method when checking for inclusion in a circle is done based upon an algorithm of Green and Sibson (1978).

We could use the grid method in all the examples given above. Note that unlike the problems dealt with in the previous subsection, avoidance problems are dynamic. We cannot afford to recompute the entire directory each time. Thus, we also need a fast method for updating the directory. For this, we will employ a dual data structure (see e.g. Aho, Hopcroft and Ullman, 1983). The operations that we are interested in are "Select a random rectangle among the good and bad rectangles", and "Update the directory" (which involves changing the status of good or bad rectangles to bad or useless rectangles, because the avoidance region grows continuously). Also, for reasons explained above, we would like to keep the good rectangles together. Assume that we have a d-dimensional table for the rectangles containing three pieces of information:

(1) The coordinates of the rectangle (usually of vector of integers, one per coordinate).

(ii) The status of the rectangle (good, bad or useless).

(iii) The position of the rectangle in the directory (this is called a pointer to the directory).

The directory is as before, except that it will shrink in size as more and more rectangles are declared useless. The update operation involves changing the status of a number of rectangles (for example, if a new circle to be avoided is added, then all the rectangles entirely within that circle are declared useless, and those that straddle the boundary are declared bad). Since we would like to keep the time of the update proportional to the number of cells involved times a constant, it is obvious that we will have to reorganize the directory. Let us use two lists again, a list of good rectangles tied down at 1 and with top at k, and a list of bad rectangles tied down at n and with top at $n-l+1$ (it has l elements). There are three situations:

(A) A good rectangle becomes bad: transfer from one list to the other. Fill the hole in the good list by filling it with the top element. Update k and l.

(B) A good or bad rectangle becomes useless: remove the element from the appropriate list, and fill the hole as in case (A). Update k or l.

(C) A bad rectangle remains bad: ignore this case.

For generation, there is only a problem when $Z>k$: when this happens, replace Z by $Z+n-l-k$, and proceed as before. This replacement makes us jump to the end of the directory.

Let us turn now to the car parking problem, to see why the grid structure is to be used with care, if at all, in avoidance problems. At first, one might be tempted to think that for fine enough grids, the performance is excellent. Also, the number of cars (N) that are eventually parked on the street cannot exceed L, the length of the street. In fact, $E(N) \sim \lambda L$ as $L \to \infty$ where

$$\lambda = \int_0^\infty e^{-2\int_0^t (1-e^{-u})/u \, du} \, dt = 0.748...$$

(see e.g. Renyi (1958), Dvoretzky and Robbins (1964) or Mannion (1964)). What determines the time of the simulation run is of course the number of uniform [0,1] random variates needed in the process. Let $\mathbf{E}$ be the event

[Car 1 does not intersect [0,1]].

Let T be the time (number of uniforms) needed before we can park a car to the left of the first car. This is infinite on the complement of $\mathbf{E}$, so we will only consider $\mathbf{E}$. The expected time of the entire simulation is at least equal to $P(\mathbf{E})E(T \mid \mathbf{E})$. Clearly, $P(\mathbf{E})=(L-1)/L$ is positive for all $L>1$. We will show that $E(T \mid \mathbf{E})=\infty$, which leads us to the conclusion that for all $L>1$, and for all grid sizes n, the expected number of uniform random variates needed is ∞. Recall however that the actual simulation time is finite with probability one.

Let W be the position of the leftmost end of the first car. Then

$$E(T \mid \mathbf{E}) = \frac{L}{L-1} \int_1^L E(T \mid W=t) \, \frac{dt}{L}$$

$$\geq \frac{L}{L-1} \int_1^{1+\frac{1}{n}} E(T \mid W=t)\, \frac{dt}{L}$$

$$\geq \frac{1}{L-1} \int_1^{1+\frac{1}{n}} \frac{1}{t-1}\, dt = \infty .$$

Similar distressing results are true for d-dimensional generalizations of the car parking problem, such as the hyperrectangle parking problem, or the problem of parking circles in the plane (Lotwick, 1984)(the circle avoidance problem of figure 3 is that of parking circles with centers in uncovered areas until the unit square is covered, and is closely related to the circle parking problem). Thus, the rejection method of Ripley (1979) for the circle parking problem, which is nothing but the grid method with one giant grid rectangle, suffers from the same drawbacks as the grid method in the car parking problem. There are several possible cures. Green and Sibson (1978) and Lotwick (1984) for example zoom in on the good areas in parking problems by using Dirichlet tessellations. Another possibility is to use a search tree. In the car parking problem, the search tree can be defined very simply as follows: the tree is binary; every internal node corresponds to a parked car, and every terminal node corresponds to a free interval, i.e. an interval in which we are allowed to park. Some parked cars may not be represented at all. The information in one internal node consists of:

p_l : the total amount of free space in the left subtree of that node;
p_r : the total amount of free space in the right subtree.

For a terminal node, we store the endpoints of the interval for that node. To park a car, no rejection is used at all. Just travel down the tree taking left turns with probability equal to $p_l/(p_l+p_r)$, and right turns otherwise, until a terminal node is reached. This can be done by using one uniform random variate for each internal node, or by reusing (milking) one uniform random variate time and again. When a terminal node is reached, a car is parked, i.e. the midpoint of the car is put uniformly on the interval in question. This car causes one of three situations to occur:

1. The interval of length 2 centered at the midpoint of the car covers the entire original interval.

2. The interval of length 2 centered at the midpoint of the car forces the original interval to shrink.

3. The interval of length 2 centered at the midpoint of the car splits the original interval in two intervals, separated by the parked car.

In case 1, the terminal node is deleted, and the sibling terminal node is deleted too by moving it up to its parent node. In case 2, the structure of the tree is

unaltered. In case 3, the terminal node becomes an internal node, and two new terminal nodes are added. In all cases, the internal nodes on the path from the root to the terminal node in question need to be updated. It can be shown that the expected time needed in the simulation is $O(L \log(L))$ as $L \to \infty$. Intuitively, this can be seen as follows: the tree has initially one node, the root. At the end, it has no nodes. In between, the tree grows and shrinks, but can never have more than L internal nodes. It is known that the random binary search tree has expected depth $O(\log(L))$ when there are L nodes, so that, even though our tree is not distributed as a random binary search tree, it comes as no surprise that the expected time per car parked is bounded from above by a constant time $\log(L)$.

3.4. Fast random variate generators.

It is known that when (X,U) is uniformly distributed under the curve of a density f, then X has density f. This could be a density in R^d, but we will only consider $d=1$ here. All of our presentation can easily be extended to R^d. Assume that f is a density on [0,1], bounded by M. The interval [0,1] is divided into N_1 equal intervals, and the interval $[0,M]$ for the y-direction is divided into N_2 equal intervals. Then, a directory is set up with k good rectangles (those completely under the curve of f), and l bad rectangles. For all rectangles, we store an integer i which indicates that the rectangle has x-coordinates $[\frac{i}{N_1},\frac{i+1}{N_1})$. Thus, i ranges from 0 to N_1-1. In addition, for the bad rectangles, we need to store a second integer j indicating that the y coordinates are $[M\frac{j}{N_2},M\frac{j+1}{N_2})$. Thus, $0 \le j < N_2$. It is worth repeating the algorithm now, because we can re-use some uniform random variates.

Generator for density f on [0,1] bounded by M

(NOTE: $D[1], \ldots, D[k+l]$ is a directory of integer-valued x-coordinates, and $Y[k+1], \ldots, Y[k+l]$ is a directory of integer-valued y-coordinates for the bad rectangles.)

REPEAT

Generate a uniform [0,1] random variate U.

$Z \leftarrow \lfloor (k+l)U \rfloor$ (Z chooses a random element in D)

$\Delta \leftarrow (k+l)U - Z$ (Δ is again uniform [0,1])

$X \leftarrow \dfrac{D[Z]+\Delta}{N_1}$

Accept $\leftarrow [Z \leq k]$

IF NOT Accept THEN

Generate a uniform [0,1] random variate V.

Accept $\leftarrow [M(Y[Z]+V) \leq f(X)N_2]$

UNTIL Accept

RETURN X

This algorithm uses only one table-look-up and one uniform random variate most of the time. It should be obvious that more can be gained if we replace the $D[i]$ entries by $\dfrac{D[i]}{N_1}$, and that in most high level languages we should just return from inside the loop. The awkward structured exit was added for readability. Note further that in the algorithm, it is irrelevant whether f is used or cf where c is a convenient constant. Usually, one might want to choose c in such a way that an annoying normalization constant cancels out.

When f is nonincreasing (an important special case), the set-up is facilitated. It becomes trivial to decide quickly whether a rectangle is good, bad or useless. Notice that when f is in a black box, we will not be able to declare a particular rectangle good or useless in our lifetime, and thus all rectangles must be classified as bad. This will of course slow down the expected time quite a bit. Still for nonincreasing f, the number of bad rectangles cannot exceed N_1+N_2. Thus, noting that the area of a grid rectangle is $\dfrac{M}{n}$, we observe that the expected number of iterations does not exceed

$$1+M\frac{N_1+N_2}{n}.$$

Taking $N_1=N_2=\sqrt{n}$, we note that the bound is $1+O(\dfrac{1}{\sqrt{n}})$. We can adjust n to off-set large values of M, the bound on f. But in comparison with strip methods, the performance is slightly worse in terms of n: in strip methods with n equal-size intervals, the expected number of iterations for monotone densities

does not exceed $1+\frac{M}{n}$. For grid methods, the n is replaced by $\sqrt{n}$. The expected number of computations of f for monotone densities does not exceed

$$\frac{l}{k+l}\left(\frac{(k+l)M}{n}\right) = \frac{lM}{n} \leq \frac{M(N_1+N_2)}{n}.$$

For unimodal densities, a similar discussion can be given. Note that in the case of a monotone or unimodal density, the set-up of the directory can be automated.

It is also important to prove that as the grid becomes finer, the expected number of iterations tends to 1. This is done below.

Theorem 3.3.

For all Riemann integrable densities f on [0,1] bounded by M, we have, as $\inf(N_1,N_2)\to\infty$, the expected number of iterations,

$$(k+l)\frac{M}{n}$$

tends to 1. The expected number of evaluations of f is $o(1)$.

Proof of Theorem 3.3.

Given an n-grid, we can construct two estimates of $\int f$,

$$\sum_{i=0}^{N_1-1} \frac{1}{N_1} \sup_{\frac{i}{N_1}\leq x\leq\frac{i+1}{N_1}} f(x),$$

and

$$\sum_{i=0}^{N_1-1} \frac{1}{N_1} \inf_{\frac{i}{N_1}\leq x\leq\frac{i+1}{N_1}} f(x).$$

By the definition of Riemann integrability (Whittaker and Watson, 1927, p.63), these tend to $\int f$ as $N_1\to\infty$. Thus, the difference between the estimates tends to 0. By a simple geometrical argument, it is seen that the area taken by the bad rectangles is at most this difference plus $2N_1$ times the area of one grid rectangle, that is, $o(1)+\frac{2M}{N_2}=o(1)$. ■

Densities that are bounded and not Riemann integrable are somehow peculiar, and less interesting in practice. Let us close this section by noting that extra savings in space can be obtained by grouping rectangles in groups of size m, and putting the groups in an auxiliary directory. If we can do this in such a way that many groups are homogeneous (all rectangles in it have the same value for $D[i]$

and are all good), then the corresponding rectangles in the directory can be discarded. This, of course, is the sort of savings advocated in the multiple table look-up method of Marsaglia (1963) (see section III.3.2). The price paid for this is an extra comparison needed to examine the auxiliary directory.

A final remark is in order about the space-time trade-off. Storage is needed for at most N_1+N_2 bad rectangles and $\frac{n}{M}$ good rectangles when f is monotone. The bound on the expected number of iterations on the other hand is $1+\frac{M}{n}(N_1+N_2)$. If $N_1=N_2=\sqrt{n}$, then keeping the storage fixed shows that the expected time increases in proportion to M. The same rate of increase, albeit with a different constant, can be observed for the ordinary rejection method with a rectangular dominating curve. If we keep the expected time fixed, then the storage increases in proportion to M. The product of storage $(1+2M/\sqrt{n})$ and expected time $(2\sqrt{n}+n/M)$ is $4\sqrt{n}+n/M+4M$. This product is minimal for $n=1, M=\sqrt{n}/2$, and the minimal value is 8. Also, the fact that storage times expected time is at least $4M$ shows that there is no hope of obtaining a cheap generator when M is large. This is not unexpected since no conditions on f besides the monotonicity are imposed. It is well-known for example that for specific classes of monotone or unimodal densities (such as all beta or gamma densities), algorithms exist which have uniformly bounded (in M) expected time and storage. On the other hand, table look-up is so fast that grid methods may well outperform standard rejection methods for many well known densities.