SIAM J. APPL. MATH.
Vol. 54, No. 4, pp. 1132–1146, August 1994

012

RANDOM GENERATION OF STOCHASTIC AREA INTEGRALS*

J. G. GAINES† AND T. J. LYONS‡

Abstract. The authors describe a method of random generation of the integrals

$$A_{1,2}(t,t+h) = \int_t^{t+h}\int_t^s dw_1(r)dw_2(s) - \int_t^{t+h}\int_t^s dw_2(r)dw_1(s),$$

together with the increments $w_1(t+h)-w_1(t)$ and $w_2(t+h)-w_2(t)$ of a two-dimensional Brownian path $(w_1(t),w_2(t))$. The method chosen is based on Marsaglia's "rectangle-wedge-tail" method, generalised to higher dimensions. The motivation is the need for a numerical scheme for simulation of strong solutions of general multidimensional stochastic differential equations with an order of convergence $O(h)$, where h is the stepsize. Previously, no method has obtained an order of convergence better than $O(\sqrt{h})$ in the general case.

Key words. stochastic differential equations, random number generation, numerical approximations

AMS subject classifications. 60H10, 65C10, 93E03

1. Introduction.

An area of current interest is the numerical solution of stochastic differential equations (SDEs)

$$dx = a(x,t)dt + \sigma_i(x,t)dw_i, \tag{1}$$

where each $w_i(t)$, $i=1,n$ is a Brownian path, x is a vector of dimension m, and the functions $a(x,t)$ and $\sigma_i(x,t), i=1,n$ take values in $\mathbf{R}^m$. Such equations arise naturally in many modelling applications, for example, in economics, population biology, particle physics, and statistics. Note that (1) has been written as an Itô sde, but what follows applies equally well to SDEs written in the Stratonovich sense.

There are essentially two types of solutions to (1), weak solutions and strong solutions. Weak solutions are used when it is the average behaviour, over all Brownian paths, of solutions or functions of solutions, that is of interest. Strong solutions are needed when an accurate picture of one particular trajectory or family of trajectories (depending on initial position or a model parameter, say) is required. Different numerical methods apply to the two types of solution. (See survey papers and books such as [13] and [5] for details.)

It is the simulation of strong solutions for SDEs that is the motivation for the present work. Although there are numerical methods that give second-order approximations to weak solutions of (1) (see Talay [14]), when it comes to strong solutions, life is much harder. In the general case of more than one dimension of noise ($n>1$) and without the commutativity condition,

$$\sigma(x,t)_i'\sigma_j(x,t) = \sigma(x,t)_j'\sigma_i(x,t) \qquad \forall x\in\mathbf{R}^m, \forall t\in\mathbf{R}^+; i,j=1,\dots,n, \tag{2}$$

where σ_i' is a matrix with elements $\sigma_i'(j,k) = \partial\sigma_{ij}/\partial x_k$, no numerical method, based only on an approximation of a Brownian path by its values at times separated by an

* Received by the editors August 12, 1992; accepted for publication (in revised form) July 22, 1993. This work was supported by EC grant SCI-0062-C.

† Department of Mathematics and Statistics, University of Edinburgh, JCMB, King's Buildings, Mayfield Road, Edinburgh EH9 3JZ, UK, (J.G.Gaines@uk.ac.ed).

‡ Department of Mathematics and Statistics, University of Edinburgh, JCMB, King's Buildings, Mayfield Road, Edinburgh EH9 3JZ, UK. Present address, Department of Mathematics, Imperial College of Science, Technology and Medicine, London SW7 2BZ, England, (T.Lyons@uk.ac.ic).

interval h, can guarantee accuracy along the trajectory of a higher order than $O(\sqrt{h})$ (see Clark and Cameron [1] for the proof).

It is known (see, e.g., [5]) that one way to obtain higher-order approximations to strong solutions of SDEs is to simulate not only increments

$$\Delta w_i(t, t+h) = w_i(t+h) - w_i(t) \tag{3}$$

along the Brownian paths, but also stochastic integrals involving the Brownian motion. To obtain accuracy of order $O(h)$, it suffices to generate and include in the numerical scheme the so-called area integrals

$$A_{i,j}(t, t+h) = \int_t^{t+h} \int_t^s dw_i(r) dw_j(s) - \int_t^{t+h} \int_t^s dw_j(r) dw_i(s), \qquad i = 1, \ldots, n; \quad j > i \tag{4}$$

However, this is much easier said than done!

This documents work resulting in a method of random generation of the integrals

$$A_{1,2}(t, t+h) = \int_t^{t+h} \int_t^s dw_1(r) dw_2(s) - \int_t^{t+h} \int_t^s dw_2(r) dw_1(s).$$

Use of these integrals allows first-order approximation of strong solutions to any SDE based on a two-dimensional Brownian path. This, although clearly only the beginning of the story, is a definite improvement. We also think that the research necessary to get us this far has uncovered various questions and suggested various techniques that are relevant to the random generation of deviates from multivariate distributions in general. (See Devroye [2] for a summary of work on multivariate distributions.)

The joint density function of $a = A_{1,2}(0,1)$, $b = \Delta w_1(0,1)$, and $c = \Delta w_2(0,1)$, using the notation defined in (3) and (4), above, is

$$f(a,b,c) = \frac{1}{2\pi^2} \int_0^\infty \frac{x}{\sinh(x)} \exp\left(\frac{-(b^2+c^2)x}{2\tanh(x)}\right) \cos(ax) dx. \tag{5}$$

(See Levy [8] and Gaveau [3].) The integral in this expression can only be calculated numerically, so there is no "quick and easy" method of generation available. The method we have chosen is based on Marsaglia's "rectangle-wedge-tail" method, generalised to higher dimensions (see Marsaglia [11] and [10] or Knuth [7] for an outline of the method).

In one dimension, Marsaglia's method involves dividing an area in $\mathbf{R}^2$ into equal rectangles and setting up tables with an entry for each rectangle. With modern computers, the amount of memory used for storage of the tables is not large. However, once we are in three dimensions and start dividing a region in $\mathbf{R}^3$ into equal pieces, the number of table entries becomes prohibitive. We have therefore been forced to a slightly more sophisticated analysis of the method to reduce storage requirements, while retaining benefits of speed. The final implementation enables us to generate the vector (a, b, c) in about 4.6 times the amount of time it takes to generate a vector of three independent numbers from a normal distribution.

2. Outline of the method.

2.1. Definition of the problem. The problem outlined above consists of generating points in $\mathbf{R}^3$ with the joint density function given in (5). However, it is possible to reduce the problem essentially to two dimensions. By setting

$$r^2 = \Delta w_1(0,1)^2 + \Delta w_2(0,1)^2$$

and $a = A_{1,2}(0,1)$ as above, we have

$$f(r,a) = \frac{r}{\pi} \int_0^\infty \frac{x}{\sinh(x)} \exp\left(\frac{-r^2 x}{2\tanh(x)}\right) \cos(ax)dx. \tag{6}$$

Therefore we have chosen the joint generation of the pair (r,a) as the central part of our method. It is then trivial to obtain correctly distributed $\Delta w_1(0,1)$ and $\Delta w_2(0,1)$ by generating θ uniformly distributed on $[0,2\pi]$ and taking

$$\Delta w_1(0,1) = r\cos\theta, \qquad \Delta w_2(0,1) = r\sin\theta.$$

See Fig. 1 for the graph of $f(r,a)$.

Note that we are generating (r,a) over unit timesteps in the first instance. Eventual scaling of a by h and r by $\sqrt{h}$ suffices to produce a sequence of points $(\Delta w_1(t,t+h), \Delta w_2(t,t+h), A_{1,2}(t,t+h))$ for any required timestep h.

Another obvious simplification consists of generating only the half of the distribution that corresponds to $a > 0$ and then giving a a random sign.

2.2. Steps in the method. Following Marsaglia's rectangle-wedge-tail method, the aim is to express the required density function $f(r,a)$ as a combination of three other densities,

$$f(r,a) = p_1 f_1(r,a) + p_2 f_2(r,a) + p_3 f_3(r,a), \tag{7}$$

where p_1, p_2, p_3 are probabilities that sum to 1, p_1 is as close to 1 as possible, and the time needed for generating numbers from the distribution corresponding to $f_1(r,a)$

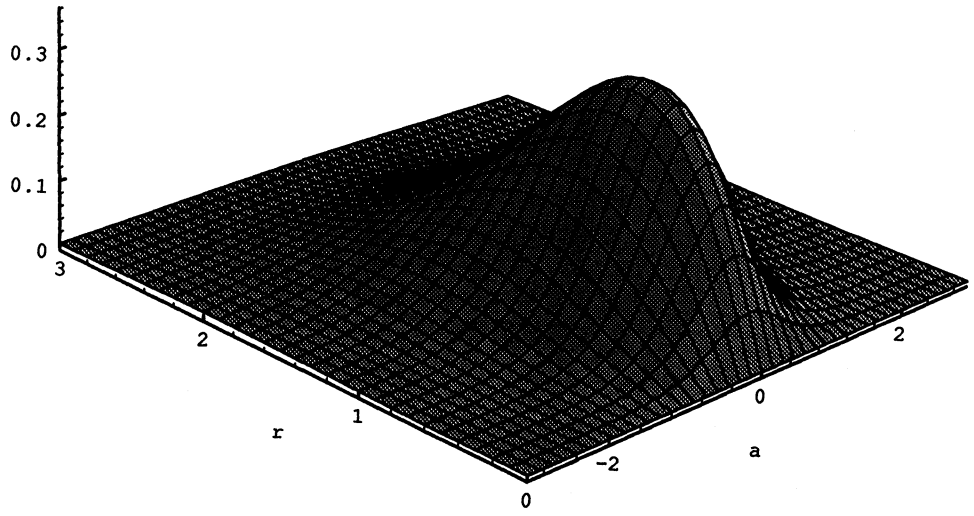

FIG. 1. *The graph of* $f(r,a)$.

is very small. The distributions corresponding to $f_2(r,a)$ and $f_3(r,a)$ may be hard to generate, but they will be used seldom enough so that the average running time for the whole routine remains acceptably low.

The volume

$$V = \{(r,a,z) | z < f(r,a), 0 \le r \le r_M, 0 \le a \le a_M\}, \tag{8}$$

where r_M and a_M are chosen suitably large, is packed with as many parallelopipeds, $B_i = \{(r,a,z) | r_i^1 < r < r_i^2, a_i^1 < a < a_i^2, z_i^1 < z < z_i^2\}$, $i = 1, \ldots, N$, as feasible. These are referred to as "boxes." (The boxes correspond to the rectangles in Marsaglia's method for generating a single random variable.) The volume under $f_1(r,a)$ is defined to be the total volume occupied by all the boxes. The "easy" density $f_1(r,a)$ is therefore a sum of uniform densities,

$$f_1(r,a) = \sum_i^N q_i \phi_i(r,a),$$

where ϕ_i is the density of points in B_i, q_i is the probability of a point (r,a) lying in box B_i (so q_i is twice the volume of B_i), and N is the number of boxes used.

The "wedges" are then the pieces left above the surface $f_1(r,a)$ and below $f(r,a)$. The "tail" is the set of points $T = \{(r,a,z) | z < f(r,a)$ and $(r > r_M$ or $a > a_M)\}$. The densities $f_2(r,a)$ and $f_3(r,a)$ are the densities of points in the wedges and in the tail, respectively.

3. The boxes. Determining the density function $f_1(r,a)$ involves first packing the volume below the surface $f(r,a)$ with boxes and then grouping the boxes in order to reduce both memory requirements and execution time of the code developed.

3.1. The packing problem. The problem is to pack the volume V, defined in (8), with boxes in such a way as to maximise the total volume occupied and satisfy the various accuracy and programming constraints imposed at later stages of the exercise.

The first decision taken is to set $r_M = a_M = 4$ in (8). This gives a volume for V of 0.49866, containing 99.732% of the distribution to be generated.

The next decision is to divide the volume under $f(r,a)$ into boxes in such a way that the dimensions of each box (length, width, and height) are all integer multiples of a chosen $l = 2^{-n}$. In this way, the volume of each box, and hence the probability of a point being in it, can be represented on the computer as a binary number using $3n$ bits, introducing no rounding errors at this stage.

A computer program was written to generate the required boxes. The volume V was subdivided using an increasingly fine mesh. As a first step, the r,a, and z axes were divided into intervals $\Delta r_1 = 2^{-n_r}$, $\Delta a_1 = 2^{-n_a}$, and $\Delta z_1 = 2^{-n_z}$. Each box defined by this mesh was tested for complete inclusion in V. The boxes found to be in V were labeled as being part of the chosen partition. In the next step, the intervals were halved, giving $\Delta r_2 = 2^{-n_r - 1}$, and so on, and the testing repeated for all boxes not already accepted. This procedure was repeated, halving the intervals at each step, until such time as a large enough fraction of the volume had been used up or the number of boxes defined was as large as thought practical. The values of n_r, n_a, n_z, the volume occupied by boxes, and the total number of boxes can be thought of as the parameters of the program. Another possible parameter, the origin of the mesh, was fixed at $(0,0,0)$ for simplicity. After experimentation, the parameter values considered acceptable were $n_r = n_a = 0$, $n_z = 6$, with a total of 13,574 boxes making up 91.19%

of the volume to be generated. This corresponds to $p_1 = 0.9119$ in (7). The numbers of boxes for each mesh size are as follows:

Step:	1	2	3	4	5
$\Delta r = \Delta a$:	1	1/2	1/4	1/8	1/16
Δz :	1/64	1/128	1/256	1/512	1/1024
Number of boxes:	3	86	519	2,431	10,535

Since the dimensions used in the final mesh were $\Delta r_5 = \Delta a_5 = 2^{-4}$, $\Delta z_5 = 2^{-9}$, 17 bits are sufficient to identify each box. The 13,574 boxes of various sizes can be broken up into 119,519 boxes of the smallest size.

Figure 2 shows all the boxes that have been allocated by the end of the third step, broken into boxes of size Δr_3 by Δa_3 by Δz_3.

Note that, rather than using a decreasing sequence of meshes, we could simply use one mesh, that with the smallest mesh size. We would obtain the same density function $f_1(r, a)$. Using the parameter values specified above, $f_1(r, a)$ would be expressed as the sum of the densities over the 119,519 boxes of smallest size. The reason for using the multistep approach is that it facilitates the next part of the exercise, which involves grouping the boxes together to form larger ones.

During this first part of the exercise, the first questions arose on the generation of multivariate random deviates using Marsaglia's method. If we fix the percentage of volume that we wish to fill with boxes, how is the number of boxes required related to the joint distribution function? Presumably, the number of boxes increases with the surface area.

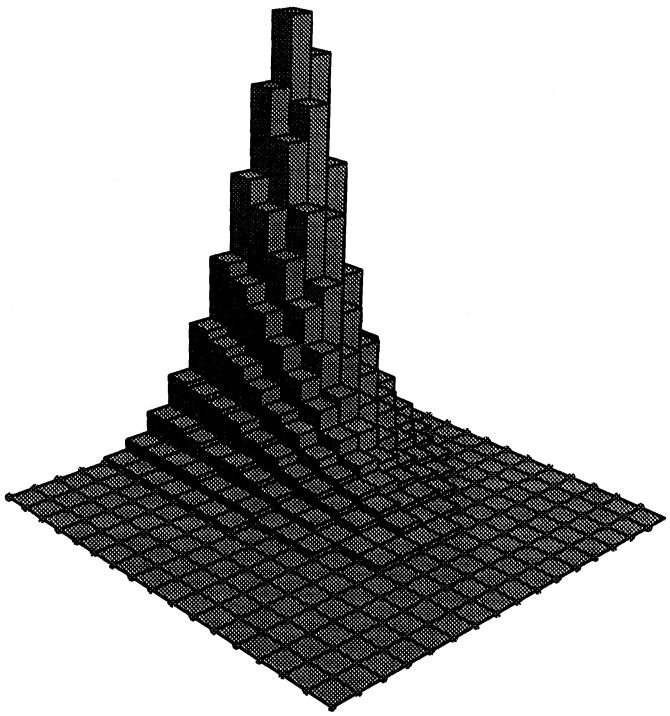

FIG. 2. *The packing at step* 3.

3.2. The entropy problem. The boxes packed under the surface $f(r, a)$ must be grouped and ordered in an efficient fashion.

Whereas in a two-dimensional problem it is currently straightforward to subdivide the required area into identical small rectangles and keep a separate record in memory for each rectangle, in a three-dimensional setting, the number of identical boxes is so large that the memory requirements would be prohibitive. In our example, 119,519 records would be required. We therefore decided to group boxes together in such a way that each subset of boxes itself formed a box.

When grouping boxes, an effort was made not only to reduce the number of boxes but also to reduce the entropy of the partition. If it were possible to use only boxes of equal volume, then a random number consisting of the right number of bits (17 in our application) would uniquely identify the correct box to use. However, once the boxes vary in volume, a set of tests is required to determine which box each point falls in and therefore which $\phi_i(r, a)$ density function to use. The tests form a binary decision tree, which can be constructed in an optimal (time-minimizing) way using Huffman's method (see [4] for the original paper or Knuth [6] for a description). This method guarantees an average number of tests performed no greater than 1 plus the entropy of the partition. The entropy was therefore taken into account when choosing a partition. The entropy can be written as

$$-\sum_{i}^{N} q_i \log_2(q_i),$$

where q_i, the probability of a point being in box B_i, is the volume of the box divided by the total volume occupied by boxes.

One possible way of grouping the boxes would be in columns, i.e.,

$$B_{i,j} = \{(r, a, z) \mid i\Delta r \le r < (i+1)\Delta r, j\Delta a \le a < (j+1)\Delta a, 0 \le z < h_{i,j}\},$$
$$i = 0 \ldots r_M/\Delta r, \qquad j = 0 \ldots a_M/\Delta a,$$

where $h_{i,j}$ is the total height of boxes piled up on the square

$$\{(r, a) | i\Delta r \le r < (i+1)\Delta r, j\Delta a \le a < (j+1)\Delta a\}.$$

With $\Delta r = \Delta a = 1/16$ and $r_M = a_M = 4$, there would be 4,096 such columns, but, if only columns with nonzero height are considered, the number reduces to 2,894. The entropy for this partition is, however, quite high, namely, 10.13 for the chosen packing.

Another method of grouping the boxes, the one that was eventually chosen, is the following. In the first instance, generate boxes of several sizes, using the method outlined in §3.1. This is equivalent to replacing groups of small boxes with larger ones. Then, in the second step, group together any boxes of the same size that are piled one on top of another. This creates pieces of columns, which are, in general, shorter than the columns described above, but many of which have larger cross sections. In this way, we reduced the number of boxes from the 13,574 in §3.1 to 2,975, giving a partition with entropy 7.14.

Once the partition has been chosen, it remains to build the decision tree needed for choosing a box with the correct probability. A computer program was written to input the partition and output the decision tree using Huffman's method. Further software details can be found in the next section.

There is no reason to believe that, given the packing, the chosen partition has minimum entropy. Designing an algorithm to generate a partition with minimum or nearly minimum entropy demands a substantial amount of further work. We plan to attempt this some time in the future and therefore to improve on the performance of the present software, perhaps extending it to higher dimensions. We conjecture that the problem of devising a partition with minimum entropy in three or more dimensions may be NP-complete. Other questions arise for general multivariate distributions, such as asking what function of the distribution the minimal entropy is.

3.3. Programming details. The part of the routine that generates points in the boxes, that is, points with the density $f_1(r, a)$, is only a few lines of executable code that relies on the data stored in one large binary tree. The tree, generated once and for all using Huffman's method, is stored in an array of records. The length of the array, equal to the number of nodes in the tree, is $2N - 1$, where N is the number of boxes (2,975). The extra $N - 1$ nodes correspond to the tests needed to determine which box to use.

Each record holds the following data: a probability, the record numbers of the left and right children, the r and a coordinates of a corner of the box, and the width of the box. (Note that, in our application, the length of each box is equal to the width. If this were not the case, the box length would also need to be given.) The records corresponding to boxes are at the ends of branches. They hold the value -1 for the record numbers of their nonexistent children. The other records, those corresponding to decisions, hold zeros in place of box coordinates and widths. This uses some extraneous memory, but produces relatively simple and fast code. On a computer with little memory, it would be possible to use two separate arrays for the decisions and the boxes. We would like to investigate the trade-off between time and memory involved.

The "probability" held at each node of the tree is cumulative and is expressed as an integer. It gives the numerator of a probability expressed as a fraction with denominator 2^n, where $n = 17$ is the number of random bits being used to determine the position.

The first step in the algorithm is to generate the n random bits, giving an integer p. If p is less than $p_B = 119,519$, where $p_B/2^n$ is the total probability of being in a box, then the point to be generated is in a box; otherwise, it is in a wedge or in the tail. If the point is in a box, then the correct box must be chosen.

The search for the correct box starts at the root of the tree. If p is less than the probability stored in the left son, then the left branch is taken; otherwise, the right branch is taken. This step is repeated until arrival at a terminal node.

The point (r, a) is then generated uniformly within the box, using the information on the position and dimensions of the box held in the node.

4. The wedges. The region in the volume V, defined in (8), and outside the union of all the boxes, can be considered a set of disjoint pieces $W = \{W_i, i = 1 \ldots n_w\}$ that, in parallel with the usage in two dimensions and for lack of a better word, we will call wedges. The density function of the set W, $f_2(r, a)$, can therefore also be expressed in terms of n_w separate functions, one for each wedge. The wedges comprise 8.55% of the whole distribution.

The packing of V with cubes, as described in §3.1, divides the square $\{(r, a, z)|z = 0, 0 \le r \le r_M, 0 \le a \le a_M\}$ into a number of identical rectangles, of sides Δr and Δa. With the values chosen of $r_M = a_M = 4$ and $\Delta r = \Delta a = 1/16$, we have $64^2 = 4,096$ base squares, upon each of which there sits a column of boxes of total

height $h_{i,j}, i, j = 1 \ldots 64$. The piece we are calling wedge is the set of points remaining above any one column and below the surface $f(r, a)$. There are therefore $n_w = 4,096$ wedges.

We chose to generate the wedges using a simple rejection method. Each wedge can be enclosed in a box

$$C_{i,j} = \{(r, a, z) | (i-1)\Delta r \leq r \leq i\Delta r, (j-1)\Delta a \leq a \leq j\Delta a, h_{i,j} \leq z \leq \tilde{f}_{i,j}\},$$

where

$$\tilde{f}_{i,j} = \max\{f(r, a), (i-1)\Delta r \leq r \leq i\Delta r, (j-1)\Delta a \leq a \leq j\Delta a\}.$$

It is then just a question of generating points uniformly distributed in one of the boxes, having chosen the box with the right probability, and testing whether the point is under the surface $f(r, a)$. In case of failure, another point is generated in the same box, and this is repeated until the test succeeds.

On average, less than half the points generated in the boxes must be rejected. However, if each test for acceptance of a point demanded evaluation by numerical integration of the function $f(r, a)$, this would be extremely costly in execution time. We have therefore chosen easy-to-calculate upper and lower approximations $f^l_{i,j}(r, a)$ and $f^u_{i,j}(r, a)$, for $f(r, a)$ on each base square $r_i \leq r \leq r_{i+1}, a_j \leq a \leq a_{j+1}$ with $r_i = i\Delta r, a_j = j\Delta a$, $i, j = 1, 64$. All points (r, a) below $f^l(r, a)$ can be accepted without having to evaluate $f(r, a)$, and, similarly, all points above $f^u(r, a)$ can be immediately rejected. This is the squeeze method, so named by Marsaglia in [9].

The approximations to $f(r, a)$ are obtained by simple interpolation. Let $f^a_{i,j}(r, a)$ be defined by

$$\begin{aligned} f^a_{i,j}(r, a) = {} & (1-t)(1-u)f(r_i, a_j) + t(1-u)f(r_{i+1}, a_j) \\ & + tuf(r_{i+1}, a_{j+1}) + (1-t)uf(r_i, a_{j+1}), \end{aligned}$$

where

$$t = (r - r_i)/(r_{i+1} - r_i), \qquad u = (a - a_j)/(a_{j+1} - a_j).$$

Then we set

$$f^l_{i,j}(r, a) = f^a_{i,j}(r, a) + \epsilon^1_{i,j}, \qquad f^u_{i,j}(r, a) = f^a_{i,j}(r, a) + \epsilon^2_{i,j}$$

with

$$\begin{aligned} \epsilon^1_{i,j} &= \min\{f(r, a) - f^a_{i,j}(r, a) | r_i \leq r \leq r_{i+1}, a_j \leq a \leq a_{j+1}\}, \\ \epsilon^2_{i,j} &= \max\{f(r, a) - f^a_{i,j}(r, a) | r_i \leq r \leq r_{i+1}, a_j \leq a \leq a_{j+1}\}. \end{aligned} \tag{9}$$

The data needed for generation of the wedges is stored in two arrays. The first array has a tree structure very similar to the data structure used for the generation of the boxes in §3.3 above. The tree is used when choosing which wedge should contain the point to be generated. The tree was again prepared using Huffman's method for reducing the average number of tests performed when finding the correct wedge.

The second array contains the information necessary for the generation of the various uniform deviates. Each entry is a pair of numbers representing the base height $h_{i,j}$ and the upper bound $\tilde{f}_{i,j}$ for a wedge. The width, length, and r and a coordinates of each box do not need to be stored, since the width Δr and length Δa are constant

over all the boxes. The use of this second array avoids some of the unnecessary use of memory mentioned in §3.3.

Two data sets are also needed for calculating the approximations $f^l_{i,j}$ and $f^u_{i,j}$. The first set consists of the function values $f(r_i, a_j)$ at the grid points, used in the interpolation. The other set contains the pairs of constants $(\epsilon^1_{i,j}, \epsilon^2_{i,j})$, as in (9), needed to ensure that the two approximations stay below and above $f(r, a)$.

5. The tail. The tail is the last, smallest, but most difficult part of the distribution. It consists of the set of points under the surface $f(r, a)$ that have $r > r_M$ or $a > a_M$. With $r_M = a_M = 4$, the tail only accounts for 0.27% of the distribution.

Yet again, the tail has been divided into a number of pieces. In each piece, points are generated using the rejection method. This time, rather than generating points from a uniform density, various nonlinear density functions have been chosen, as being reasonably good approximations to $f(r, a)$ in the regions considered. In the table below, Table 1, details are given of the regions into which the tail has been divided and the density functions used. The rate reported in the last column indicates the average number of points generated to find one point in the relevant region. The points in the last region, which has probability less than one in a million, are not generated with the right distribution.

TABLE 1

Region	r_1	r_2	a_1	a_2	Probability	Comparison function	Rate
R_1	4	12	0	4	$2.98 \cdot 10^{-4}$	$\exp(-r^2/2)$	2.3
R_2	4	8	4	8	$3.65 \cdot 10^{-5}$	$3/2\exp(-r^2/2 - 3a^2/2r^2)$	2.0
R_3	2	4	4	8	$1.80 \cdot 10^{-3}$	$\exp(-\frac{\pi}{2}a)$	2.6
R_4	0	0.5	4	6	$1.61 \cdot 10^{-6}$	$15r\exp(-\pi a)$	2.6
R_5	0.5	1	4	6	$1.96 \cdot 10^{-5}$	$15r\exp(-2.8a)$	2.8
R_6	1	1.5	4	6	$1.20 \cdot 10^{-4}$	$25r\exp(-2.6a)$	3.0
R_7	1.5	2	4	6	$3.86 \cdot 10^{-4}$	$25r\exp(-2.4a)$	3.2
R_8	1	2	6	8	$6.57 \cdot 10^{-6}$	$40r\exp(-2.4a)$	4.2
R_9	2	5	8	10	$4.15 \cdot 10^{-6}$	$0.7\exp(-\frac{\pi}{2}a)$	2.4
	Remainder				$3.81 \cdot 10^{-7}$		

The various regions in the tail are depicted in Fig. 3.

6. Performance. The results of time tests are laid out in Table 2. The time taken to generate $(\Delta w_1(0,h), \Delta w_2(0,h), A_{1,2}(0,h))$ triples is compared with the time taken to generate triples of uniform deviates and to generate triples of normally distributed random numbers using an application of Marsaglia's method. The same pseudo-random uniform number generator was used throughout. All the code was written in C and run on a Sun IPC Sparc workstation.

TABLE 2

Number of triples	Time taken		
	Uniform deviates	Normal deviates	(w_1, w_2, a)
10,000	<1s	<1s	3s
100,000	2s	6s	28s
1,000,000	15s	1min1s	4min42s

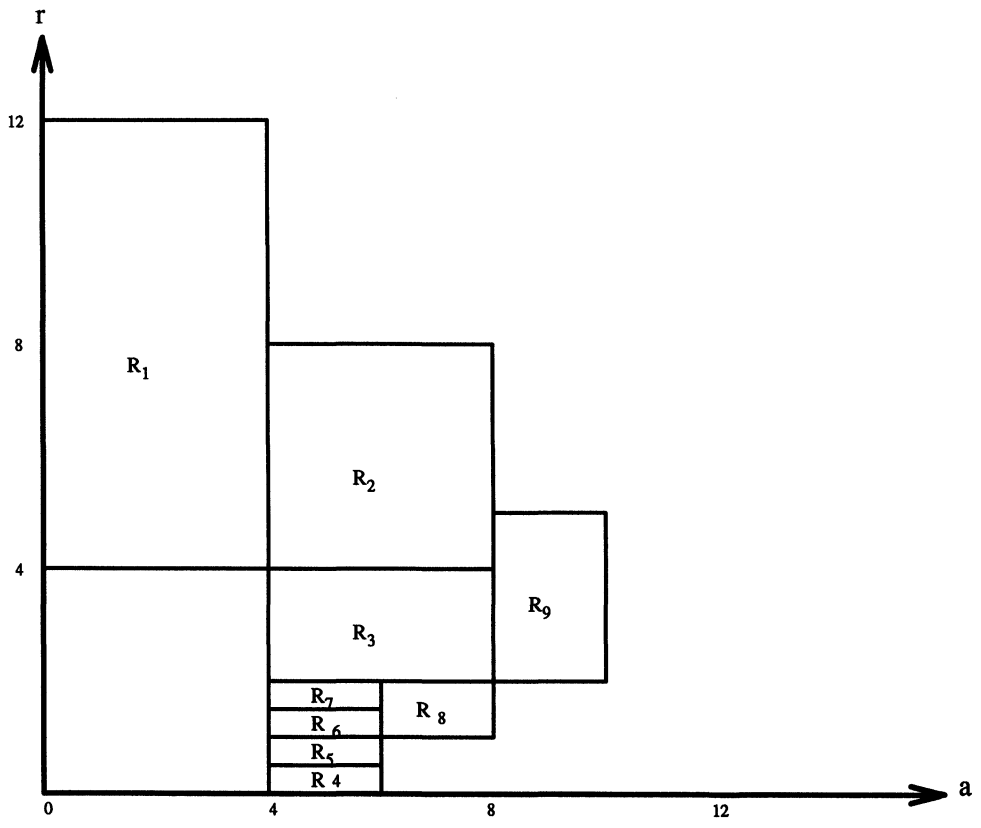

FIG. 3. *The regions in the tail.*

The generation of area integrals therefore takes between four and five times as long as the generation of increments along the Brownian path. If we were to obtain a strong solution of order $O(h)$ by subdividing the timesteps and generating increments of the Brownian path over steps of length $h' < h$, the condition on the size of h' would be $h' \leq h^2$. For $h < 1/4$, we obtain $h > 4h'$. So, for any reasonably small stepsize h, it is much quicker to obtain a solution of order $O(h)$ by generation of the $A_{1,2}(t, t+h)$ integrals than by generating $\Delta w_1(t, t+h)$ and $\Delta w_2(t, t+h)$ over smaller steps.

7. An example of application. As an illustration, we have used area integrals generated by our method in the numerical solution of the bilinear Itô sde

$$dx = Ax\ dw_1 + Bx\ dw_2, \tag{10}$$

where A and B are constant 2×2 real matrices. The matrices A and B can be reduced to one of several canonical forms, in which case they depend on four real parameters. For example, we can take

$$A = \begin{bmatrix} a & 0 \\ 0 & -a \end{bmatrix}, \qquad B = \begin{bmatrix} b & c \\ c & d \end{bmatrix}.$$

This may be considered as a fundamental example, since locally any SDE can be considered linear.

The ease of numerical solution of (10) depends on the stability of the system and hence on the parameters a, b, c, d. When the system is stable, it is possible to obtain a good approximate solution with a discretisation scheme that does not involve the area integrals using quite large timesteps. However, when the system is unstable, it can become impossible to obtain a good solution without including the area integrals in the discretisation scheme, unless extremely small timesteps are used (sometimes too small to be feasible on a computer).

We show what happens in a "difficult" case. We have taken $a = c = 2$ and $b = d = 1$ and compared the results using three different discretisation schemes. The approximate solutions given by these schemes all converge to the solution of the SDE (1) taken in the Itô sense (and can only be used for a Stratonovich SDE if it is first converted into an Itô SDE). The first scheme, scheme A, is the well-known Euler–Maruyama scheme

$$\tilde{x}^{k+1} = \tilde{x}^k + a(\tilde{x}^k)h + \sigma_i(\tilde{x}^k)\Delta w_i^k,$$

where h is the chosen timestep, $\tilde{x}^k$ is the approximation obtained to x at time kh, and we define

$$\Delta w_i^k = \Delta w_i\,[kh, (k+1)h]$$

(see (3) in §1). The second scheme, scheme B, is

$$\begin{aligned}\tilde{x}_i^{k+1} = \tilde{x}_i^k + \left[a_i(\tilde{x}^k) - \tfrac{1}{2}\tfrac{\partial \sigma_{ij}}{\partial x_r}(\tilde{x}^k)\sigma_{rj}(\tilde{x}^k)\right]h \\ + \sigma_{ij}(\tilde{x}^k)\Delta w_j^k + \tfrac{1}{2}\tfrac{\partial \sigma_{ip}}{\partial x_r}(\tilde{x}^k)\sigma_{rj}(\tilde{x}^k)\Delta w_j^k \Delta w_p^k,\end{aligned}$$

and scheme C, the Milshtein scheme (see [5]), is the same as the previous scheme, with added terms involving the area integrals, given by

$$\begin{aligned}\tilde{x}_i^{k+1} = \tilde{x}_i^k + \left[a_i(\tilde{x}^k) - \frac{1}{2}\frac{\partial \sigma_{ij}}{\partial x_r}(\tilde{x}^k)\sigma_{rj}(\tilde{x}^k)\right]h + \sigma_{ij}(\tilde{x}^k)\Delta w_j^k \\ + \frac{1}{2}\frac{\partial \sigma_{ij}}{\partial x_r}(\tilde{x}^k)\sigma_{rj}(\tilde{x}^k)\left(\Delta w_j^k\right)^2 \\ + \frac{1}{2}\sum_{j<p}\frac{\partial \sigma_{ip}}{\partial x_r}(\tilde{x}^k)\sigma_{rj}(\tilde{x}^k)\left[\Delta w_j^k \Delta w_p^k - A_{j,p}(kh, (k+1)h)\right] \\ + \frac{1}{2}\sum_{j>p}\frac{\partial \sigma_{ip}}{\partial x_r}(\tilde{x}^k)\sigma_{rj}(\tilde{x}^k)\left[\Delta w_j^k \Delta w_p^k - A_{p,j}(kh, (k+1)h)\right].\end{aligned}$$

The Euler–Maruyama scheme converges to the true solution with rate $O(\sqrt{h})$ in the general multidimensional set up. Scheme B converges with the same rate, but is asymptotically efficient (see [12]); i.e., the leading coefficient of the variance of the error in each step is minimal.

In Table 3 we present the approximate value obtained for x_1 at time $t = 5$ for one particular simulation of the Brownian path, using each of the three discretisation schemes and using a succession of step sizes ranging from $h = 2^{-4}$ to $h = 2^{-16}$.

When the timestep is greater than approximately 2^{-8}, the results obtained without area integrals, that is, using scheme A or scheme B, are completely wrong after

TABLE 3

	$\tilde{x}_1(5)$		
$-\log_2(h)$	Scheme A	Scheme B	Scheme C
4	2.447207	−0.017475	1.251238
5	−0.111236	0.030939	3.066002
6	0.008536	0.074882	4.243542
7	0.642300	1.281455	4.759604
8	1.096747	1.903669	3.751482
9	4.383487	3.538671	3.430095
10	5.292467	3.883448	3.430359
11	4.034335	3.030876	3.483880
12	3.704273	2.955552	3.460112
13	2.879655	2.705867	3.428692
14	3.036604	3.062775	3.431992
15	3.330327	3.167863	3.425553
16	3.415980	3.377057	3.424928

about $t = 3$. This is illustrated in the form of a graph in Fig. 4, depicting the three approximate trajectories obtained for x_1 using a timestep $h = 2^{-6}$ and an accurate solution, obtained by using scheme C and $h = 2^{-16}$.

It is only once that the timestep is small enough that it makes sense to measure the difference between the various approximations. In Fig. 5 we show the differences between the approximate solutions for x_1 obtained using each of the three discretisation schemes with a timestep of $h = 2^{-10}$ and the same accurate solution used previously.

To obtain the same accuracy using scheme A or scheme B as using scheme C, it is necessary to square the timestep and therefore the number of timesteps. If, when using scheme A, the Euler–Maruyama scheme, we wish to obtain the same accuracy as with scheme C, using area integrals, with the stepsize $h = 2^{-8}$, then we must

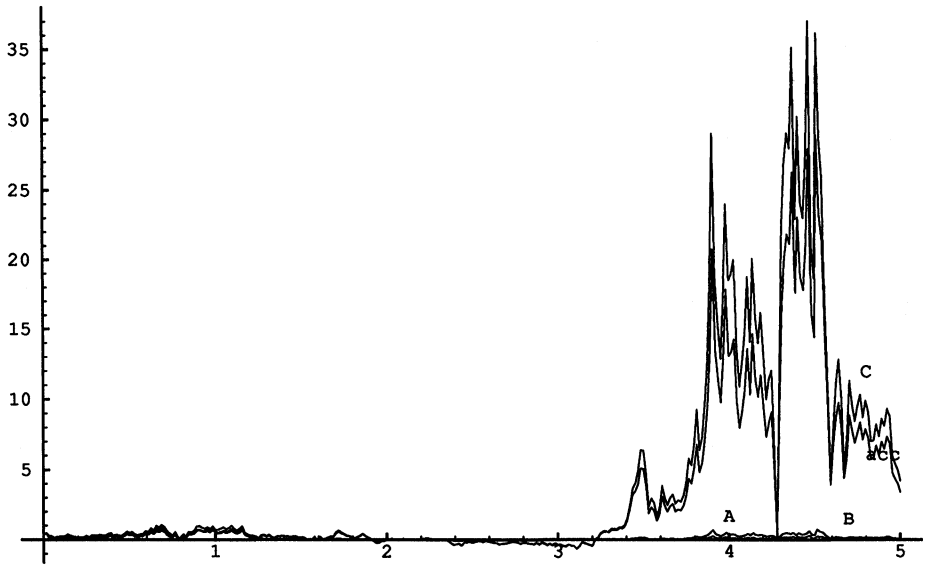

FIG. 4. *Approximations with and without area integrals.*

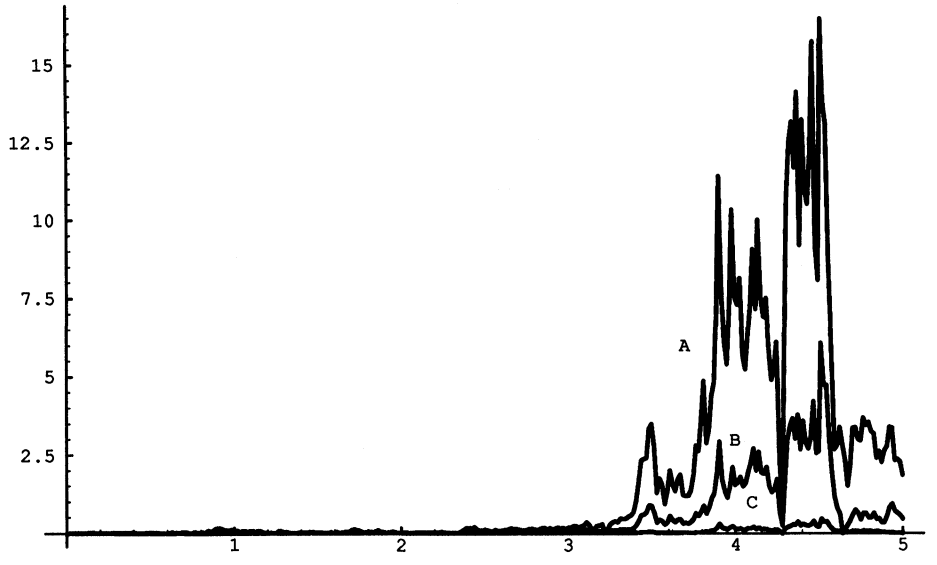

FIG. 5. *Differences between approximations and an accurate solution.*

take $h = 2^{-16}$, hence multiplying the number of steps by 256. The computing time is not increased by a factor of 256, since more function evaluations are performed per timestep using scheme C than by using scheme A. Timed simulations using (10) give a factor of 160. On a Sun Sparc station, it took roughly three minutes to perform 1,000 simulations with scheme C and $h = 2^{-8}$ and eight hours for the same number of simulations with scheme A and $h = 2^{-16}$.

This shows that, if all we want is to perform one single simulation, then using a simple scheme and small timesteps is not going to occupy much computing time, but, if we wish to do many simulations, varying, say, the starting point or the parameter values, then the time factor will become very important. (A million different simulations would take 50 hours or 8,000 hours, to obtain the same accuracy with and without area integrals.) The time factor would clearly be increased further if scheme C were replaced by a Runge–Kutta-type discretisation scheme, still using area integrals but not involving evaluation of derivatives at each timestep.

In the future, we foresee that the use of area integrals when simulating strong solutions to SDEs will become as automatic as the use of random numbers from a normal distribution is today. After all, once a good routine has been developed and implemented in numerical libraries, the ordinary user will only need to call this routine from each program and will not need to be concerned with the details of how the routine works.

8. Conclusion. This is, of course, just the tip of the iceberg. On one hand, to obtain a strong numerical solution of order $O(h)$ to a stochastic differential equation dependent on a Brownian path of dimension $n > 2$, not just one set of integrals may be needed, but several, and the sets are all correlated. In the most general case, we would need to generate the $n(n+1)/2$ correlated random variables

$$\Delta w_i(t, t+h),\ A_{i,j}(t, t+h), \qquad i = 1, n; \quad j < i.$$

It is theoretically possible to reduce the problem to the generation of n correlated random variables and one random element of $\mathbf{O}_n$, which can be generated in n steps. Let A be the matrix $A_{ij} = \int_0^1 \int_t^s dw_i(r)dw_j(s) - \int_0^1 \int_t^s dw_j(r)dw_i(s)$. An orthogonal matrix $S \in \mathbf{O}_n$ can be chosen so that $S^t w = \bar{w}$ and $S^t A S = \bar{A}$ with

$$\bar{w} = \begin{pmatrix} \bar{w}_1 \\ 0 \\ \bar{w}_2 \\ 0 \\ \vdots \\ \bar{w}_m \\ 0 \end{pmatrix}, \qquad \bar{A} = \begin{pmatrix} 0 & \lambda_1 & 0 & 0 & \dots & 0 & 0 \\ -\lambda_1 & 0 & 0 & 0 & \dots & 0 & 0 \\ 0 & 0 & 0 & \lambda_2 & \dots & 0 & 0 \\ 0 & 0 & -\lambda_2 & 0 & \dots & 0 & 0 \\ \vdots & & & & & & \\ 0 & 0 & 0 & 0 & \dots & 0 & \lambda_m \\ 0 & 0 & 0 & 0 & \dots & -\lambda_m & 0 \end{pmatrix},$$

if $n = 2m$, or

$$\bar{w} = \begin{pmatrix} \bar{w}_1 \\ 0 \\ \bar{w}_2 \\ 0 \\ \vdots \\ \bar{w}_m \\ 0 \\ \bar{w}_{m+1} \end{pmatrix}, \qquad \bar{A} = \begin{pmatrix} 0 & \lambda_1 & 0 & 0 & \dots & 0 & 0 \\ -\lambda_1 & 0 & 0 & 0 & \dots & 0 & 0 \\ 0 & 0 & 0 & \lambda_2 & \dots & 0 & 0 \\ 0 & 0 & -\lambda_2 & 0 & \dots & 0 & 0 \\ \vdots & & & & & & \\ 0 & 0 & 0 & 0 & \dots & 0 & \lambda_m \\ 0 & 0 & 0 & 0 & \dots & -\lambda_m & 0 \\ 0 & 0 & 0 & 0 & \dots & 0 & 0 \end{pmatrix},$$

if $n = 2m + 1$. Therefore, if we can obtain the joint distribution function of

$$\bar{v} = \{\bar{w}_1, \dots, \bar{w}_m, \lambda_1, \dots, \lambda_m\}$$

from the joint distribution of

$$v = \{w_1, \dots, w_n, A_{12}, A_{13}, \dots, A_{n-1,n}\},$$

we can then generate the vector v and a random matrix $S \in \mathbf{O}_n$ and simply recover w and A by setting $w = S\bar{w}$ and $A = S\bar{A}S^t$.

We would also like to derive and implement the bridge that would allow us to generate sets of points $(\Delta w_1(t_n, t_{n+1}), \Delta w_2(t_n, t_{n+1}), A_{1,2}(t_n, t_{n+1}))$ over steps of length $h/2$ given the corresponding set for stepsize h. All this will demand a lot more work and imagination.

On the other hand, by demonstrating that Marsaglia's method for speedy random number generation can be applied to multivariate distributions, we have raised general questions (see the end of §§3.1 and 3.2) on the time and memory constraints involved.

If readers wish to obtain either a complete copy of the routine we have written to generate area integrals or else a copy of the tables of numbers used by the routine (unfortunately too long to be given here), they are welcome to contact one of the authors.

REFERENCES

[1] J. M. C. Clark and R. J. Cameron, *The maximum rate of convergence of discrete approximations for stochastic differential equations*, in Stochastic Differential Systems, B. Grigelionis, ed., Lecture Notes in Control and Information Sciences No. 25, Springer-Verlag, Berlin, 1980.

[2] L. DEVROYE, *Non Uniform Random Variate Generation*, Springer-Verlag, Berlin, New York, 1986.

[3] B. GAVEAU, *Principe de moindre action, propagation de la chaleur et estimées sous elliptiques sur certains groupes nilpotents*, Acta Math., 139 (1977), pp. 95–153.

[4] D. A. HUFFMAN, *A method for the construction of minimum redundancy codes*, Proc. IRE, 40 (1952), pp. 1098–1101.

[5] P. E. KLOEDEN AND E. PLATEN, *Numerical Solution of Stochastic Differential Equations*, Applications of Mathematics, Vol. 23, Springer-Verlag, Berlin, New York, 1992.

[6] D. E. KNUTH, *Fundamental Algorithms*, The Art of Computer Programming, Vol. 1, Addison-Wesley, Reading, MA., 1973.

[7] ———, *Seminumerical Algorithms*, The Art of Computer Programming, Vol. 2, Addison-Wesley, Reading, MA, 1973.

[8] P. LÉVY, *Wiener's random function, and other Laplacian random functions*, in Second Berkeley Symposium on Mathematical Statistics and Probability, J. Neyman, ed., 1951.

[9] G. MARSAGLIA, *The squeeze method for generating gamma variates*, Comput. Math. Appl., 3 (1977), pp. 321–325.

[10] G. MARSAGLIA, K. ANANTHARAYANAN, AND N. J. PAUL, *Improvements on fast methods for generating normal random variables*, Inform. Process. Lett., 5 (1976), pp. 27–30.

[11] G. MARSAGLIA, M. D. MACLAREN, AND T. A. BRAY, *A fast procedure for generating normal random variables*, Comm. Assoc. Comput. Mach., 7 (1964), pp. 4–10.

[12] N. J. NEWTON, *An asymptotically efficient difference formula for solving stochastic differential equations*, Stochastics, 19 (1986), pp. 175–206.

[13] E. PARDOUX AND D. TALAY, *Discretization and simulation of stochastic differential equations*, Acta Appl. Math., 3 (1985), pp. 23–47.

[14] D. TALAY, *Second order discretization schemes of stochastic differential systems for the computation of the invariant law*, Stochastics Stochastics Rep., 29 (1990), pp. 13–36.