Proceedings of the 1986 Winter Simulation Conference
J. Wilson, J. Henriksen, S. Roberts (eds.)

CORRELATION INDUCTION WITHOUT THE INVERSE TRANSFORMATION

Bruce Schmeiser
School of Industrial Engineering
Purdue University
West Lafayette, IN 47907, U.S.A.

Voratas Kachitvichyanukul
Deparment of Industrial and Management Engineering
The University of Iowa
Iowa City, IA 52242, U.S.A.

ABSTRACT

Inducing correlation between estimators is a common way to try to reduce variance in simulation experiments. To induce the correlation between estimators, random variates are generated as functions of the same random-number streams. Although the optimal correlation induction occurs with the inverse transformation, the inverse can be quite slow compared to other methods for generating random variates. We discuss an approach for generating random variates quickly while still obtaining substantial correlation induction.

1. INTRODUCTION

We consider two topics that each have received considerable attention in the simulation literature, but which seldom have been considered together: inducing dependence and efficient random-variate generation. Specifically, we consider how to induce dependence among inputs to a simulation experiment via modifications of existing fast algorithms.

We develop here algorithms in which substantial correlation can be induced, yet which have execution speeds almost as fast as the original algorithms from which they arise. The modifications required are oriented toward the monotonicity and synchronization properties usually associated with obtaining correlated random variates. The resulting algorithms are useful in distribution sampling experiments in which random variate generation time comprises the bulk of the computing effort. Correlation induction in discrete-event simulation, for which random-generation time is usually negligible, is usually better pursued with the inverse transformation, as discussed below and, e.g., in Bratley, Fox and Schrage (1983).

The organization of this paper is as follows. In Section 2, we briefly discuss the background of correlation induction, emphasizing the usual approach via the inverse transformation. In Section 3, we discuss an approach for maintaining synchronization and partial monotonicity in order to obtain correlation induction via fast algorithms. Section 4 contains three examples showing how existing algorithms can be modified.

2. BACKGROUND

The ideas in this paper combine ideas of correlation induction and efficient random-variate generation. We discuss each in turn.

2.1 Correlation Induction

By *correlation induction* we mean creating dependence between intermediate estimators in such a way that a combined grand estimator has smaller variance than if the same combination of independent intermediate estimators were used to obtain the grand estimator. The use of such correlation typically arises in the variance-reduction contexts of common random numbers, antithetic variates, and external control variates, as discussed in most simulation textbooks. Wilson (1984) and Nelson (1986) are surveys of variance reduction with up-to-date citations.

Let $Z_i^{(M)}$ denote the i^{th} intermediate estimator, $i = 1, 2, \cdots, i*$, for $M = D, I$, where D denotes dependence and I denotes independence. Let F_i denote the common marginal distribution of $Z_i^{(D)}$ and $Z_i^{(I)}$. Assume the $Z_i^{(I)}$s are mutually independent; i.e., their joint distribution function is $F^{(I)} = \prod_{i=1}^{i*} F_i$. For both $M = D$ and $M = I$, let $Z^{(M)} = c(Z_1^{(M)}, \ldots, Z_{i*}^{(M)})$ denote a combined grand estimator, where the function c is the same for both the dependent and independent cases. Then the difference in distribution between $Z^{(D)}$ and $Z^{(I)}$ arises entirely from the difference in the dependency structure of $F^{(D)}$ and $F^{(I)}$. Changing this dependency structure from

independence to various forms of dependency underlie correlation induction.

Consider the three most common examples:

1. A typical application of common random numbers is in the comparison of two systems, where $Z_{2i-1}^{(M)}$ is the intermediate estimator from the i^{th} simulation run of the first system and $Z_{2i}^{(M)}$ is the intermediate estimator from the i^{th} simulation run of the second system. Then $c = (i*/2)^{-1} \sum_{i=1}^{i*/2} [Z_{2i-1}^{(M)} - Z_{2i}^{(M)}]$ is an estimator of the mean difference between the systems. If $\text{Corr}\{Z_{2i-1}^{(D)}, Z_{2i}^{(D)}\} > 0$ and $\text{Corr}\{Z_{2i-1}^{(D)}, Z_j^{(D)}\} = 0$ for $j \neq 2i$, then $\text{Var}\{Z^{(D)}\} < \text{Var}\{Z^{(I)}\}$.

2. A typical application of antithetic variates is in the evaluation of a single system, where $F_i^{(M)}$ are all identical, each resulting from one simulation run of the system. Then $c = i*^{-1} \sum_{i=1}^{i*/2} [Z_{2i-1}^{(M)} + Z_{2i}^{(M)}]$ is an estimator of the system parameter of interest. If $\text{Corr}\{Z_{2i-1}^{(D)}, Z_{2i}^{(D)}\} < 0$, and $\text{Corr}\{Z_{2i-1}^{(D)}, Z_j^{(D)}\} = 0$ for $j \neq 2i$, then $\text{Var}\{Z^{(D)}\} < \text{Var}\{Z^{(I)}\}$.

3. A typical application of external control variates is in the evaluation of a single system aided by the simulation of an auxiliary system, with $Z_{2i-1}^{(M)}$ being the i^{th} simulation run of the system of interest and $Z_{2i}^{(M)}$ being the i^{th} simulation run of the auxiliary system. Then $c = [(i*/2)^{-1} \sum_{i=1}^{i*/2} Z_{2i-1}^{(M)}] - b [\sum_{i=1}^{i*/2} Z_{2i}^{(M)}/(i*/2) - \text{E}\{Z_i^{(M)}\}]$, where b is a constant or itself a function of the $Z_i^{(M)}$s. If $\text{Corr}\{Z_{2i-1}^{(D)}, Z_{2i}^{(D)}\} > 0$, $\text{Corr}(Z_{2i-1}^{(D)}, Z_j^{(D)}) = 0$ for $j \neq 2i$, and $b > 0$, then often $\text{Var}\{Z^{(D)}\} < \text{Var}\{Z^{(I)}\}$.

Given that dependence among the intermediate estimators is sometimes useful, the question is how to best induce this dependence. In simulation, the answer is that the $U(0,1)$ random-number streams are manipulated to create (one hopes) correlated input random variates (e.g., interarrival times and service times), which in turn create (one hopes) correlated outputs (e.g., customer wait times), which in turn create correlated intermediate estimators (e.g., average wait time for entire runs). If the correlations are of the appropriate sign and (in some cases) of sufficient magnitude, the variance of the grand estimator is reduced.

The maximal correlation induction that can be induced between two intermediate estimators with marginal distribution functions F_i and F_j is obtained by the inverse transformation $Z_i = F_i^{-1}(U)$ and $Z_j = F_j^{-1}(U)$, where $U \sim U(0,1)$. Similarly, the minimal (that is, most negative) correlation is obtained by the inverse transformation $Z_i = F_i^{-1}(U)$ and $Z_j = F_j^{-1}(1-U)$, where $U \sim U(0,1)$.

This optimal property of the inverse transformation has led to its central role in correlation induction. In trying to approach this optimality with other methods, we need to consider the two properties that the inverse transformation obtains automatically: synchronization and monotonicity. For the inverse transformation, synchronization is the property that the same random number U_i is used to obtain the same Z_i. Monotonicity is the property that Z_i is a monotonic function of U_i.

Typically the inverse transformation of the intermediate estimator, F_i, is not known explicitly. Attention therefore focuses on being able to generate pairs of correlated input random variates, in the hope that the model will transform these correlated values in such a way (almost monotonically, in some sense) that they lead to correlated intermediate estimators. To generate correlated input random variates, the same random numbers should be used to generate the correlated random variates and the use of the random numbers should be oriented toward obtaining as close to a monotonic transformation as possible between one of the random numbers and the random variate generated. Correlation, and in turn variance reduction, degrades as either synchronization or monotonicity degrades.

The success of the ideas in Section 3 rests on the ability of the developers of random-variate generators to obtain near synchronization and near monotonicity.

2.2 Random-Variate Generation

In addition to using the inverse transformation $X = F_X^{-1}(U)$ to generate random realizations of X, algorithms based on composition, acceptance/rejection, and/or special properties can be used. Schmeiser (1980) discusses random-variate generation with emphasis on these four underlying concepts. Devroye (1986) is an excellent up-to-date encyclopedic text on random-variate generation.

As we have seen, the advantage of using the inverse transformation is the automatic synchronization and monotonicity obtained in the generation of random-variate inputs. The disadvantage is that the inverse transformation can be quite slow and can incur numerical problems, especially when the inversion is not closed form. Although Chen and Asau (1974), Fishman and Moore (1984), and

Ahrens and Kohrt (1981) have discussed methods for speeding the inversion, for most distributions noninversion algorithms are substantially faster, especially when set-up time is considered.

A typical state-of-the-art algorithm uses composition at some level to allow the algorithm to return a random variate very quickly a large fraction of the time. The tails of the distribution are often treated separately. More than one random number, and often a random number of random numbers (a phrase we enjoy), is used to return one random variate. The result is often an algorithm that destroys both synchronization of random numbers and monotonicity.

The reason then becomes clear why simulation software often provides both the inverse transformation and a fast routine: The inverse transformation is used when correlation induction is needed (and sometimes for generation of order statistics and generation from truncated distributions) and fast methods are used in other cases.

3. CONCEPTS

In this section we discuss concepts underlying algorithm development when the goal is to create fast algorithms that obtain good correlation induction. The resulting algorithms are based on state-of-the-art algorithms, but are a bit slower and usually obtain less correlation induction than the inverse transformation.

Consider an arbitrary distribution and a fast algorithm for generating random variates from it. If the fast algorithm is the inverse transformation, there is no problem, so assume the algorithm uses one or more of composition, acceptance/rejection, or special properties. (Acceptance/complement we treat as a special case of composition.) The problem is to modify the algorithm's use of $U(0,1)$ random numbers to obtain positive or negative correlation as desired. The concepts to be developed here are how to maintain some degree of synchronization and some degree of monotonicity.

3.1 Synchronization

If the algorithm uses a constant number of random numbers per variate, as can occur with composition and special properties, synchronization is no problem. Assume the algorithm uses a random number of random numbers due to acceptance/rejection logic. We obtain synchronization via the use of two random-number streams. The first stream is used for the first iteration of acceptance/rejection. If the value is rejected on the first iteration, future iterations use the second random-number stream. We hope to obtain highly correlated random variates when returned on the first iteration and at worst zero-correlated random variates when returned on later iterations.

The two random-number streams require passing two seeds to the algorithm, say seed1 and seed2. Let n denote the maximal number of random numbers needed per iteration. The logic is then

```
   subroutine randx (seed1, seed2, parameters, x)
   dimension u(n)
   . . .
   do 10 i=1,n
10 call random (seed1,u(i))
   go to 40
20 do 30 i=1,n
30 call random (seed2,u(i))
40 perform accept/reject logic to get a candidate x
   if accept, return x
   go to 20
   ...
   end
```

In many algorithms the number of random numbers needed per iteration can be made constant.

Since in state-of-the-art algorithms the probability of acceptance is high, the lack of synchronization of stream two is not crucial. If the correlation between variates arising from the second stream were zero, then the correlation obtained from the algorithm would be the product of the correlation between variates obtained on the first iteration and the probability that both variates are obtained on the first iteration. For an acceptance/rejection algorithm, the correlation from the first iteration is that of the inverse transformation of the majorizing distribution.

However, as we see in Section 4.3, the correlation between variates arising from the second steam (that is, after the first iteration) is not zero, since the second stream feeds the same numbers to the generator, although on different calls. Such correlation in the example is of the correct sign. Therefore we create the antithetic effect in the second stream as well as the first.

3.2 Monotonicity

Given the above or some other method of synchronizing many of the random numbers, we still must provide some form of monotonicity. The approach taken here is to select

one random number and modify the algorithm to make it as much a monotonic function of the generated random variate as possible. The other random numbers are coded not to mimic the random variate but to aid the relationship between the selected random number and the random variate.

First consider composition. Here the distribution to be generated (which is either the distribution of interest or one arising as a majorizing function) is written as a mixture of subdistributions. Often two of the subdensities correspond to the tails of the distribution. One random number, say w, is used to select a particular subdistribution. A big step toward monotonicity is to map low values of w to the left tail, high values of w to the right tail, and intermediate values to the body of the distribution. If the subdistributions correspond to disjoint intervals on the x axis, then the values of w are assigned to the subdistributions in increasing order. The code is then

```
      subroutine randx (seed1, seed2, parameters, x)
c        p: composition probabilities, ordered on x
c        m: number of subdistributions
      dimension p(m)
      call random (seed1, w)
      j=0
  100 j = j+1
      if (w .lt. p(j))  go to 200
      w = w - p(j)
      go to 100
  200 generate x from subdistribution j
      return
```

If the intervals are not disjoint, sometimes it is possible to combine subdistributions to make them disjoint (e.g., Kachitvichyanukul, Cheng, and Schmeiser (1985). When the composition yields no disjoint intervals, the algorithm is not a good candidate, as for example the noncentral chi-square distribution written as an infinite mixture of chi-square distributions.

Now consider special properties. Here x is generated as a deterministic function of a fixed number of other random variates, $x_1, \cdots, x_p$, each of which is generated from the $U(0,1)$ random numbers. Consider $p = 1$, as in generating a lognormal random variate from a normal random variate or a Weibull random variate from an exponential random variate. Monotonicity is most easily retained in the first case by flipping the normal on its mean and in the second case by flipping the random number in the inverse transformation of the exponential. For any value of p, if X is a symmetric random variable, antithetic values can be obtained by flipping it around its mean. The best procedure is not so obvious in other cases. Our experience is that good correlations are obtained by choosing one of the random variates, say X_1, to make as correlated with X as possible. This correlation is a surrogate for the usual monotonicity condition. The choice of a single variable for obtaining correlation is related to ideas in Cheng (1982).

4. EXAMPLES

We discuss three examples in this section. The first illustrates how blindly flipping random numbers can lead to a good algorithm for correlation induction. The second illustrates the well-known phenomenon of obtaining zero (or negligible) correlation when blindly flipping random numbers in an attempt to obtain antithetic random variates. The third is an illustration of a useful algorithm obtained by modifying an existing fast algorithm using the concepts developed in Section 3.

4.1 Example 1: Triangular Random Variates

We first consider an example for which almost any attempt at correlation will be successful. Within a composition algorithm, we often need to obtain a symmetric triangular random variable x with mean zero and width $2d$. Typically the variate is generated by the special property that the difference of two $U(0,d)$ random variates; i.e., $x = d \times (u - v)$. Here $p = 2$, but X is symmetric, so obtaining the antithetic value by flipping x is straightforward, as illustrated in the first code below. The second code is for flipping both u and v, which is algebraically equivalent since $x = d \times ((1-u)-(1-v)) = d \times (v-u) = -x$. (The slightly slower version $x = d \times (u+v-1)$ seems in some sense more monotonic since x is monotonically increasing with respect to both u and v, but the correlation induced is the same as with $u-v$, since the direction of the monotonicity is irrelevant.) The third code shown is for the sometimes suggested correlation induction approach of switching random-number streams, which in this case again is algebraically equivalent to the first two codes.

(flip x)

```
      subroutine randx (seed1, seed2, parameters, x, ianti)
      ...
      u = random(seed1)
      v = random(seed1)
      x = u - v
      if (ianti .gt. 0)  x = -x
      ...
      end
```

```
(flip random numbers)
      subroutine randx (seed1, seed2, parameters, x, ianti)
      ...
      u = random(seed1)
      v = random(seed1)
      if (ianti .le. 0)  go to 1
      u = 1 - u
      v = 1 - v
    1 x = u - v
      ...
      end


(swap streams)
      subroutine randx (seed1, seed2, parameters, x, ianti)
      ...
      s1 = seed1
      s2 = seed2
      if (ianti .le. 0)  go to 1
      s1 = seed2
      s2 = seed1
    1 x = random(s1) + random(s2)
      ...
      end
```

4.2 Example 2: Normal-Variate Generation

The classic example of special properties is the Box-Muller method for generating normal random variates. As in the first example, synchronization is no problem since exactly two random numbers are required to obtain two normal random variate. Unlike the first example, blind application of random-number switching or random-number stream does not necessarily result in the desired correlation induction. Consider the following code.

```
      subroutine norbm (dseed, x, IANTI)
c     bruce schmeiser  voratas kachitvitchyanukul
c       september 1986  purdue university
c     ggubfs is the IMSL U(0,1) generator
      double precision dseed
      data l/0/
      if (l .gt. 0) go to 100
      l = 1
      u = ggubfs (dseed)
      v = ggubfs (dseed)
      s = sqrt (-2.*alog(u))
      IF (IANTI .LE. 0)  GO TO  10
      V = V + .5
      IF (V .GT. 1.) V = V -1.
   10 a = 2 * 3.14159 * v
      x = s * sin (a)
      x2 = s * cos (a)
      return
  100 x = x2
      l = 0
      return
      end
```

When called twice with the same seed, but once with IANTI=1 and once with IANTI=0, this subroutine returns standard normal random variates with correlation −1. Note that the two-to-one mapping of random numbers to random variates makes the idea of maintaining a monotonic mapping less straightforward. Correlation is obtained by not changing the first random-number stream and using rotation sampling on the second stream (i.e., $v = v + 1 (\text{mod } 1)$). Simply flipping all random-number streams, $u = 1-u$ and $v = 1-v$, for the antithetic run results in zero correlation. Flipping only the first stream also yields zero correlation. Flipping only the second stream yields a perfect positive correlation; that is, the antithetic run is algebraically the same as the first run, the worst possible case. Switching random-number streams produces zero correlation.

While a nice example for illustrative purposes, the Box-Muller algorithm and more generally the normal distribution is not a useful example, since we can easily obtain negative correlation by simply setting $x = -x$ in the antithetic run.

4.3 Example 3: Beta-Variate Generation

In this subsection we provide a more useful example by modifying the beta-variate generator B2PE (Schmeiser and Babu (1980)) to obtain correlated observations. B2PE, which is valid for beta parameters $p > 1$ and $q > 1$, is not as fast as the more complicated algorithm B4PE also described in Schmeiser and Babu (1980), but is representative of the types of problems we face in modifying existing algorithms to induce correlation. If we had used a more complicated algorithm, we think the induced correlations would be greater and we know the modification would be more difficult. Note: In step two of B2PE in Schmeiser and Babu (1980), the definition of λ_2 should involve upper case P and Q rather than lower case.

Another reason a beta random-variate generation algorithm is a nice example is that linearly transformed normal and gamma (including Erlang, chi square, and exponential) random variables are limiting cases. Therefore, this example demonstrates the concepts of this paper over a wider range of distributions than one might first think.

Before discussing in detail the modified algorithm, denoted B2PEA, we summarize briefly our empirical experience. In the symmetric cases $p = q$, the induced correlation is −1, the same as the inverse transformation. In the limiting case as the beta goes to the exponential, the algorithm again creates obtains the minimum correlation, which is a little less than −.6 for exponential random

variables. In other cases the induced correlation is not as good as that obtained by the inverse transformation, although over wide ranges of the parameters more than 80% of the optimal correlation obtained. We have not tested the positive correlation obtained between two different beta distributions.

One seeming disadvantage of using noninversion algorithms to generate beta variates is that cross correlations are created. In particular, if (X_i, X_i') denotes the i^{th} generated pair, B2PE causes Corr $(X_i, X_j') \neq 0$ even for $j \neq i$. But some thought shows that such correlation does not invalidate the experiment, since Corr $(X_i, X_j) = 0$ and Corr $(X_i', X_j') = 0$ for all $i \neq j$. Better still, our experimental evidence is that the cross correlations tend to improve Corr $(\overline{X}, \overline{X}')$. For example, when $p = 300$ and $q = 30$ the modified algorithm B2PEA induces Corr $(X_i, X_i') \simeq -.7$ and Corr $(\overline{X}, \overline{X}') \simeq -.8$. Thus for the very specific (and unnecessary) experiment of using Monte Carlo experimentation to estimation E $\{X\}$, more negative correlation than first thought is induced between the antithetic intermediate estimators $\overline{X}$ and $\overline{X}'$. Since most estimators are not linear functions of the random variate inputs, the correlation induced between intermediate estimators of interest will be different from $-.8$. Nevertheless, it is encouraging that such cross correlations seem to help rather than hinder estimation.

The original algorithm B2PE is listed in Figure 1. The modified algorithm B2PEA is listed in Figure 2, with the modifications shown in upper case. We comment below on each important change.

1. New parameters DSEED2 and IANTI are added, with meanings as defined in the code comments.

2. The five lines beginning with line number 55 are the key to synchronization. Each iteration of the acceptance/rejection algorithm requires exactly two random numbers, u and v. On the first iteration, the random numbers are drawn from the first random-number stream. On the second and following iterations, the random numbers are drawn from the second random-number stream, beginning at line 5. Synchronization is maintained for the first stream and not maintained for the second stream. But since the probability of the first iteration ending in acceptance is large, the lack of synchronization of the second stream is relatively inconsequential.

3. Line 6 flips the first random number, u, when indicated by IANTI. The intent of the modifications is to obtain monotonicity between u and x. Despite the lack of synchronization for stream two, we still flip u on all iterations, since this can create the useful cross correlations discussed above.

4. The logic between lines 6 and 7 is now for the left tail, whereas previously it was for the body of the distribution. The body was checked first previously since it had the largest probability of being chosen. The left tail is now checked first in the attempt to obtain monotonicity. By chance, very little code needed to be changed for the left tail; only the method of obtaining a new random number from the old changed: $u = u/p_2$.

5. The logic for the body of the distribution lies between lines 7 and 8. Again the only renormalization is that of u, which is embedded in the definition of x: $x = x_2 + (u - p_2) = x_2 + (x_4 - x_2) * (u - p_2)/(p_1 - p_2)$.

6. The right-tail logic still lies between line numbers 8 and 9. Since the position of the right-tail logic did not change, the renormalization of u is unchanged. The definition of x is slightly changed, now taking the logarithm of $1.-u$ rather than u to obtain monotonicity. The substitution of $1.-u$ here causes the same substitution in the next two lines.

ACKNOWLEDGMENT

Michael R. Taaffe participated in useful discussions of concepts and James J. Swain provided comments that improved the presentation.

REFERENCES

Ahrens, J.H. and Kohrt, K.D. (1981). "Computer Methods for Efficient Sampling from Largely Arbitrary Statistical Distributions," *Computing* **26**, 19-31.

Bratley, P., Fox, B.L. and Schrage, L.E. (1983). *A Guide to Simulation.* Springer-Verlag, New York.

Chen, H.C. and Asau, Y. (1974). "On Generating Random Variates from an Empirical Distribution," *AIIE Transactions* **6**, 163-166.

Cheng, R.C.H. (1982). "The Use of Antithetic Variates in Computer Simulations," *Journal of the Operational Research Society* **33**, 229-237.

```
      subroutine b2pe (dseed, pa, qa, x)
c     bruce schmeiser  voratas kachitvichyanukul
c       september 1986   purdue university
c     to generate one beta random variate
c        schmeiser, b.w. and babu, a.j.g. (1980)
c          "beta variate generation via exponential
c           majorizing functions," operations
c           research 28, 917-926.
      double precision dseed
      data psave/-1./
      if (pa .eq. psave .and. qa .eq. qsave)  go to 5
c..... initialization
      psave = pa
      qsave = qa
c     ..... ensure p .le. q (for computer arithmetic)
      p = pa
      q = qa
      if (pa .le. qa)  go to 2
      p = qa
      q = pa
c     ..... calculate constants as a function of p and q
    2 pp = p - 1.
      qq = q - 1.
      r = pp + qq
      s = r * alog(r)
      x2 = 0.
      f2 = 0.
      f4 = 0.
      x3 = pp/r
      x4 = 1.
      if (r .le. 1.)  go to 4
      d = sqrt (pp*qq/(r-1.)) / r
      if (d .ge. x3)  go to 3
c     ..... left tail
      x2 = x3 - d
      xl2 = p/x2 - q/(1.-x2)
      f2 = exp (pp*alog(x2/pp) + qq*alog((1.-x2)/qq) + s)
    3 if (x3 + d .ge. 1.)  go to 4
c     ..... right tail
      x4 = x3 + d
      xl4 = qq/(1.-x4) - pp/x4
      f4 = exp (pp*alog(x4/pp) + qq*alog((1.-x4)/qq) + s)
c     ..... region areas
    4 p1 = x4 - x2
      p2 = p1
      if (xl2 .gt. 0.)  p2 = f2 / xl2 + p1
      p3 = p2
      if (xl4 .gt. 0.)  p3 = f4 / xl4 + p2
c..... generate random variates
    5 u = ggubfs(dseed) * p3
      v = ggubfs(dseed)
    6 if (u .gt. p1)  go to 7
c     ..... middle region
      x = x2 + u
      if (x .lt. x3 .and. v .lt. f2 + (x-x2)*(1.-f2)/(x3-x2))  go to 10
      if (x .ge. x3 .and. v .lt. f4 + (x4-x)*(1.-f4)/(x4-x3))  go to 10
      go to 9
    7 if (u .gt. p2)  go to 8
c     ..... left tail region
      u = (u-p1)/(p2-p1)
      x = x2 + alog(u)/xl2
      if (v .lt. (xl2*(x-x2)+1.)/u)  go to 10
      if (x .le. 0.)  go to 5
      v = v*f2*u
      go to 9
c     ..... right tail region
    8 u = (u-p2)/(p3-p2)
      x = x4 - alog(u)/xl4
      if (v .lt. (xl4*(x4-x)+1.)/u)  go to 10
      if (x .gt. 1.)  go to 5
      v = v*f4*u
c     ..... final rejection compares for all three regions
    9 a = alog(v)
      if (a .gt. -((x-x3)**2)*(r+r))  go to 5
      if (a .gt. pp*alog(x/pp) + qq*alog((1.-x)/qq) + s)  go to 5
   10 if (pa .gt. qa)  x = 1. - x
      return
      end
```

Figure 1: FORTRAN code for the original algorithm B2PE.

```
      subroutine b2pea (dseed, DSEED2, pa, qa, x, IANTI)
c     bruce schmeiser  voratas kachitvichyanukul
c       september 1986   purdue university
c     to generate one beta random variate
c       this code is a modification of b2pe, described in
c         schmeiser, b.w. and babu, a.j.g. (1980)
c           "beta variate generation via exponential
c            majorizing functions," operations
c            research 28, 917-926.
c     the modification allows correlated random variates
c         positive correlation
c           (for common random numbers or control variates):
c             ianti = 0 always
c         negative correlation (for antithetic variates):
c             ianti = 0 for run one.  ianti = 1 for run two.
c     arguments:
c       input:
c         dseed:  seed for the first random-number stream
c         dseed2: seed for the second random-number stream
c         pa:     first parameter
c         qa:     second parameter
c         ianti:  indicator to control correlation
c       output:
c         dseed:  the new seed for the first stream
c         dseed2: the new seed for the second stream
c         x:      the generated beta random variate
c       called subprograms:
c         ggubfs:  the imsl U(0,1) random-number generator
      double precision dseed, DSEED2
      data psave/-1./
      if (pa .eq. psave .and. qa .eq. qsave)  GO TO 55
c..... initialization
c     ..... save argument values for later calls
      psave = pa
      qsave = qa
c     ..... ensure p .le. q (for computer arithmetic)
      p = pa
      q = qa
      if (pa .le. qa)  go to 2
      p = qa
      q = pa
c     ..... calculate constants as functions of p and q
    2 pp = p - 1.
      qq = q - 1.
      r = pp + qq
      s = r * alog(r)
      x2 = 0.
      f2 = 0.
      f4 = 0.
      x3 = pp/r
      x4 = 1.
      if (r .le. 1.)  go to 4
      d = sqrt (pp*qq/(r-1.)) / r
      if (d .ge. x3)  go to 3
c     ..... left tail
      x2 = x3 - d
      xl2 = pp/x2 - qq/(1.-x2)
      f2 = exp (pp*alog(x2/pp) + qq*alog((1.-x2)/qq) + s)
    3 if (x3 + d .ge. 1.)  go to 4
c     ..... right tail
      x4 = x3 + d
      xl4 = qq/(1.-x4) - pp/x4
      f4 = exp (pp*alog(x4/pp) + qq*alog((1.-x4)/qq) + s)
c     ..... region areas
    4 P2 = 0.
      IF (XL2 .GT. 0.)  P2 = F2 / XL2
      P1 = (X4-X2) + P2
      P3 = P1
      IF (XL4 .GT. 0.)  P3 = F4 / XL4 + P1
c..... generate random variates
   55 U = GGUBFS(DSEED) * P3
      v = ggubfs(dseed)
      GO TO 6
    5 U = GGUBFS(DSEED2) * P3
      V = GGUBFS(DSEED2)
    6 IF (IANTI .GT. 0)  U = P3 - U
      IF (U .GE. P2)  GO TO 7
c     ..... left tail region
      U = U / P2
      X = X2 + ALOG(U)/XL2
      if (v .lt. (xl2*(x-x2)+1.)/u)  go to 10
      if (x .le. 0.)  go to 5
      v = v*f2*u
      go to 9
c     ..... middle region
    7 IF (U .GT. P1)  GO TO 8
      X = X2 + (U-P2)
      if (x .lt. x3 .and. v .lt. f2 + (x-x2)*(1.-f2)/(x3-x2))  go to 10
      if (x .ge. x3 .and. v .lt. f4 + (x4-x)*(1.-f4)/(x4-x3))  go to 10
      go to 9
c     ..... right tail region
    8 U = (U-P1)/(P3-P1)
      X = X4 - ALOG(1.-U)/XL4
      IF (V .LT. (XL4*(X4-X)+1.)/(1.-U))  GO TO 10
      if (x .ge. 1.)  go to 5
      V = V*F4*(1.-U)
c     ..... final rejection compares for all three regions
    9 a = alog(v)
      if (a .gt. -((x-x3)**2)*(r+r))  go to 5
      if (a .gt. pp*alog(x/pp) + qq*alog((1.-x)/qq) + s)  go to 5
   10 if (pa .gt. qa)  x = 1. - x
      return
      end
```

Figure 2: FORTRAN code for the modified algorithm B2PEA.

Devroye, Luc (1986). *Non-Uniform Random Variate Generation.* Springer-Verlag, New York.

Fishman, G.S. and Moore, L.R. III (1984). "Sampling from a Discrete Distribution while Preserving Monotonicity," *The American Statistician* **38**, 219-223.

Kachitvichyanukul, V., Cheng, S-W.J. and Schmeiser, B. (1985). "Fast Poisson and Binomial Algorithms for Correlation Induction," Technical Report 86-1, Department of Industrial and Management Engineering, The University of Iowa.

Nelson, B.L. (1986). "A Perspective on Variance Reduction in Simulation Experiments," Working Paper Series 1985-011, Department of Industrial and Systems Engineering, The Ohio State University.

Schmeiser, B.W. (1980). "Random Variate Generation: A Survey." In: *Simulation with Discrete Models: A State-of-the-Art View,* Volume 2 of *Proceedings of the Winter Simulation Conference,* (T.I. Ören, C.M. Shub, and P.F. Roth, eds.). IEEE, New York, 79-104.

Schmeiser, B.W. and Babu, A.J.G. (1980). "Beta Variate Generation Via Exponential Majorizing Functions," *Operations Research* **28**, 917-926.

Wilson, J.R. (1984). "Variance Reduction Techniques for Digital Simulation," *American Journal of Mathematical and Management Sciences* **4**, 277-312.

AUTHOR'S BIBLIOGRAPHIES

BRUCE SCHMEISER is a professor in the School of Industrial Engineering at Purdue University. He received his undergraduate degree in Mathematical Sciences and master's degree from the Department of Industrial and Management Engineering at The University of Iowa. His Ph.D. is from the School of Industrial and Systems Engineering at the Georgia Institute of Technology.

Dr. Schmeiser has served on the editorial boards of *IIE Transactions, Communications in Statistics, B: Simulation and Computation, Journal of Quality Technology, American Journal of Mathematical and Management Sciences,* and the *Handbook of Industrial Engineering.* He is the past chairman of the TIMS College on Simulation and Gaming. He represents ORSA on the Winter Simulation Conference Board of Directors, currently serving as Vice-Chairman.

Dr. Schmeiser's research interests are the probabilistic and statistical aspects of digital-computer stochastic simulation. He has published articles on input modeling, random-variate generation, output analysis, and variance reduction.

Bruce Schmeiser
School of Industrial Engineering
Purdue University
West Lafayette, IN 47907, U.S.A.
(317) 494-5422
schmeise@gb.ecn.purdue.edu

VORATAS KACHITVICHYANUKUL is an assistant professor in Industrial and Management Engineering at The University of Iowa. His current research interests are development of integrated simulation environments, special purpose simulation languages, simulation with microcomputers, and random variate generation.

Dr. Kachitvichyanukul holds a B.S. degree in Chemical Engineering from National Taiwan University, a Master of Engineering in Industrial Engineering and Management from Asian Institute of Technology, and a Ph. D. in Industrial Engineering from Purdue University. He is member of the Society for Computer Simulation, Institute of Industrial Engineers, The Institute of Management Sciences, and the Association for Computing Machinery.

Voratas Kachitvichyanukul
Department of Industrial and Management Engineering
The University of Iowa
Iowa City, IA 52242, U.S.A.
(319) 353-4848
aegvorwy%ulamvs.bitnet@wiscvm.wisc.edu