

The Monte Carlo Method in SPECT

.....

David P. Lewis, Ph.D.

Dept. of Biomedical Engineering, University of North Carolina, Chapel Hill, NC 27599 USA

The Monte Carlo (MC) method is a computer technique for simulating phenomena which have random events. It is named after the city in Monaco famous for its gambling casinos. In nuclear medicine, radioactive decay, photon scatter in the body and in the collimator, and photon absorption in the crystal are all random events and can be modelled with the MC method. This paper is an introduction to the MC method with applications to SPECT. We start by motivating the problem of simulating the process of SPECT scanning. We then introduce the MC method by giving a simple example of how MC can be used to estimate the value of π . Next we describe methods for generating random numbers on a computer, a very important part of MC. Finally we discuss the use of MC in SPECT.

Why simulate SPECT?

Various physical effects degrade SPECT images, including attenuation, scatter, detector response, and noise⁽¹⁾. In order to improve SPECT images it is important to know how to compensate for these effects⁽¹⁾. For example, iterative reconstruction using attenuation compensation has been shown to improve quantitative accuracy in SPECT. However, it is still unclear whether attenuation compensation improves clinical diagnosis.

There are several reasons why we simulate SPECT imaging. First, with computer simulation we know the phantom exactly. After we reconstruct the simulated projection data, we can compare the reconstructed image with the known phantom. In this way we can test the reconstruction technique. For example, if we reconstruct the projection data with different reconstruction techniques, we can determine the best technique by comparing the recon-

structed images with the phantom. With patients we do not know the true activity distribution, so comparing reconstruction techniques with patient data is more difficult. With a phantom experiment we can also know the true activity distribution, but not as accurately as we can with computer simulation.

Second, with simulation we can know how scatter contributes to the projection data. This is not possible to know with experimental or clinical data, since the camera cannot distinguish between primary and scatter photons. (Although collimation and energy windowing allow us to reject some of the scatter photons, some scatter photons are still detected and can have a significant effect on image quality and quantitative accuracy.) In fact, with simulation we can separate the primary and scatter components of the projection data; in fact, we can separate the scatter component into first-order scatter, second-order scatter, etc. This helps us to study

how scatter degrades images as well as to devise means for compensating for scatter.

Third, with computer simulation we can see how changes to the imaging hardware or to the data acquisition protocols will affect the images. For example, one can try out different collimator designs in simulation without actually constructing anything. As another example one can use simulation to acquire projection data using different energy windows and then determine which energy window is best for a given application.

Fourth, the “patient” population can be controlled in order to focus on a particular disease state to study. For example, if one is studying the detection of septal defects in cardiac SPECT then one can create some phantoms which have the defect and some which do not. Moreover, one can also control the size of the defect. In SPECT, patient motion can degrade the image, making diagnosis more difficult. With simulation, motion is not typically present. As a result, one can use simulation to study effects which might otherwise be obscured by patient motion. On the other hand, through simulation one can study how different types of patient motion, such as cardiac beating, breathing, or gross patient motion can affect SPECT imaging.

Fifth, one can systematically study various physical effects by isolating them in simulation. For example, suppose that one wanted to study how detector response blurring affects lesion detection. Simulation can be performed with no detector response blurring—i.e. perfect collimation—as well as different amounts of detector response blurring (corresponding to different collimators). This cannot be

done in practice. One could also generate projection data with perfect energy resolution or with extremely low noise, for example.

Since SPECT is governed by stochastic processes, it is natural to use Monte Carlo for simulation. Monte Carlo programs in nuclear medicine are fairly complicated, so we introduce the method with a simple example.

Simple example of Monte Carlo — estimating π

In this example we present a simple method for estimating π with MC. Although this is a very inefficient way to estimate π , it illustrates some important features of the MC method. For comparison, the true value of π to 6 decimal places is 3.141593.

Consider a circle inscribed in a square, as shown in Figure 1. Now suppose that we were to drop balls onto the square randomly, so that each ball has an equal probability of landing anywhere on the square. The probability that a ball will land in the circle is $\pi/4$, since the ratio of the area of the circle to the area of the square is $\pi/4$. This means that if we drop many balls onto the square, the fraction of balls landing in the circle is approximately $\pi/4$.

We can easily perform this experiment on a computer. For concreteness, we choose to use a square which has sides of length 2, and the inscribed circle is the unit circle. To simulate a ball dropping randomly on the square, we need to pick two random numbers corresponding to the x- and y-coordinates. This means that we need a way to generate many random numbers between -1 and +1. We will discuss the generation of random numbers shortly.

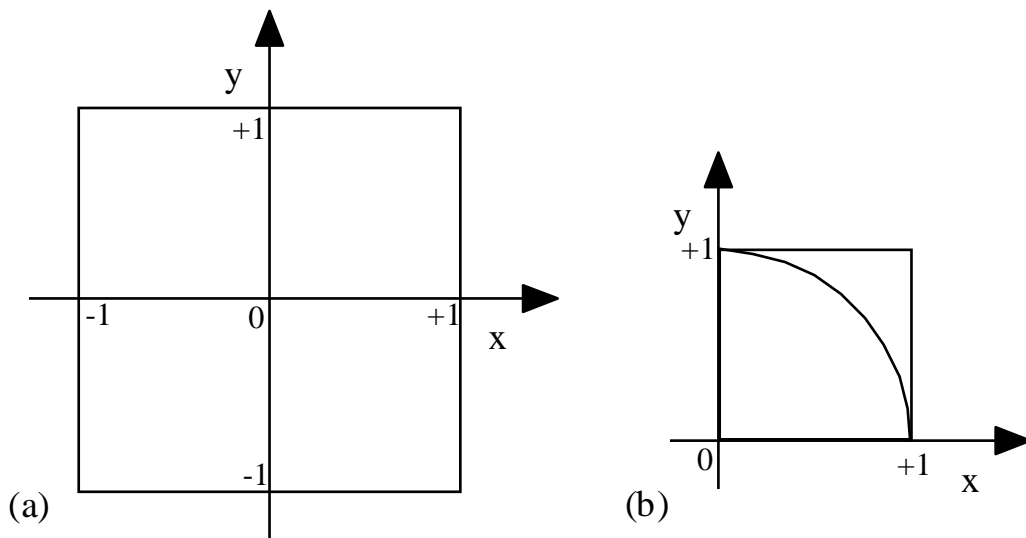


Fig. 1 (a) Circle inscribed in a square for the simple MC method for estimating π . The area of the circle divided by the area of the square is $\pi/4$. For concreteness we have chosen to use a square with sides of length 2. (b) To make the simulation even simpler we use just the first quadrant.

After generating the numbers x and y , we test to see if they correspond to a point inside the circle, i.e. if $x^2 + y^2 < 1$. We repeat the process many times, counting the number of times a ball lands in the circle. The procedure is illustrated in the pseudocode in Figure 2. Also, note that the problem is symmetric, so we can choose to use only one quadrant. In that case we need to generate random numbers between 0 and 1. We did this for the simulations, and we present the results below.

```

initialize: num_in = 0, r = 1
choose N
for n = 1 to N {
  generate random number (x,y)
  if sqrt(x2 + y2) < r,
    num_in <= num_in + 1
}
 $\pi$ _estimate = 4·num_in / N

```

Table 1 shows the results of simulating 10 balls, with the estimate of π after each ball has been dropped. A total of 8 balls landed in the circle, giving an estimate of 3.2 for π — obviously a poor estimate! Clearly we need to simulate a very large number of balls in order to get an accurate estimate of π . Table 2 shows the estimate of π after various numbers of balls have been dropped. It can be seen that the estimate converges slowly towards π as the number of balls increases.

Fig. 2 Pseudocode for program simulating the experiment for estimating π . The variables are: r = radius of circle, N = number of balls to drop, n = counting variable, num_in = number of balls which have fallen inside the circle.

Table 1 Results of simulating 10 balls

n	x	y	in circle?	total inside	estimate of π
1	0.106056	0.562682	x	1	4.000000
2	0.597442	0.452214	x	2	4.000000
3	0.113265	0.925333	x	3	4.000000
4	0.994239	0.865538		3	3.000000
5	0.965050	0.071491	x	4	3.200000
6	0.787452	0.133523		5	3.333333
7	0.973945	0.504676		5	2.857143
8	0.016421	0.763225	x	6	3.000000
9	0.201433	0.327989	x	7	3.111111
10	0.327925	0.673846	x	8	3.200000

Since a very large number of balls are needed to estimate π accurately, it certainly would not be feasible to perform a real experiment to estimate π by dropping balls onto a square! However, on a computer the simulation of 1,000,000 balls, for example, is very fast. On a DEC Personal Workstation 500AU computer it took about 1 second to simulate 1,000,000 balls.

Random number generation on a computer

It is not possible to generate truly random numbers on a computer. Instead, computers generate *pseudorandom* numbers^(2,3). These numbers are generated with numerical algorithms and appear random only to those who do not know the algorithm. For example, the x- and y-coordinates in Table 1 have been generated by a numeri-

cal algorithm and thus are not truly random, but they appear random to us. In this paper we will not distinguish between truly random numbers and pseudorandom numbers but will use the terms interchangeably. A computer algorithm which generates pseudorandom numbers is called a *random number generator* (RNG). Random number generators are at the heart of Monte Carlo programs so we describe them here.

A simple random number generator

The first RNG, devised by von Neumann during World War II, is called the mid-square method^(2,4). The user chooses a number between 0 and 1 with four digits, for example 0.7914. This first number is called the *seed*. (In most other RNG's the seed is an integer.) This number is then squared and the middle four digits are taken to give the first random number in the sequence. In our example, $0.7914^2 = 0.62631396$. The middle four digits are 6313, so the first random number generated is 0.6313. To get the next random number we repeat the process: $0.6313^2 = 0.39853969$, so the second random number is 0.8539. Then $0.8539^2 = 0.72914521$, so the third random number is 0.9145. This procedure is repeated every time a new

Table 2 Estimates of π for different numbers of balls dropped (N)

N	est. of π
1	4.000000
10	2.800000
100	2.880000
1,000	3.120000
10,000	3.136800
100,000	3.139000
1,000,000	3.141524

Table 3 Random numbers generated with the mid-square method for a seed of 0.9876

i	random #		i	random #	i	random #
seed	0.9876	...	70	0.8120	80	0.2916
1	0.5353	...	71	0.9343	81	0.5030
2	0.6546	...	72	0.2916	82	0.3009
3	0.8501	...	73	0.5030	83	0.0540
4	0.2669	...	74	0.3009	84	0.2916
5	0.1235	...	75	0.0540	85	0.5030
6	0.5252	...	76	0.2916	86	0.3009
7	0.5835	...	77	0.5030	87	0.0540
8	0.0472	...	78	0.3009	88	0.2916
9	0.2227	...	79	0.0540	89	0.5030

random number is needed. The seed, which is chosen only once, determines the sequence of pseudorandom numbers generated. If a different seed is chosen, a different sequence of pseudorandom numbers will be generated. Table 3 shows a sequence of pseudorandom numbers generated with the mid-square method.

Although the numbers in Table 3 appear random, the mid-square method is a very poor random number generator. First, if the number 0.0000 occurs in the sequence, all of the subsequent numbers will be 0.0000. If the number 0.2500 occurs, all of the subsequent numbers will be 0.2500. Second, there are only 10,000 distinct numbers that can be generated with this method, since

only 4 digits are important. At some point the sequence will repeat itself. In Table 3, it can be seen that some point the sequence enters a cycle with a small period: the 76th number is the same as the 72nd, the 77th is the same as the 73rd, etc. Third, the mid-square method does not produce numbers uniformly distributed between 0 and 1. In fact, most of the numbers it generates are less than 0.5! Figure 3 is a histogram showing how frequently the pseudorandom numbers are in the ranges 0-0.1, 0.1-0.2, ..., 0.9-1.0. The seed was 0.5678 and 10,000 pseudorandom numbers were generated. For an ideal random number generator, the histogram should be flat.

Column 2 of Table 4 shows results of using

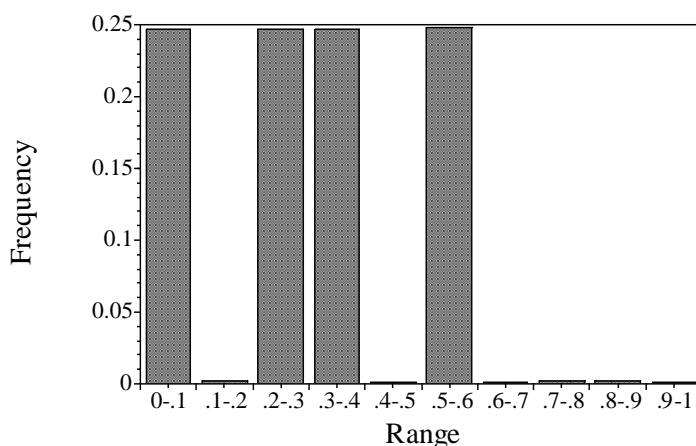


Fig. 3 Histogram showing the frequency of pseudorandom numbers occurring in different ranges for the mid-square method. Seed = 0.5678 and 10,000 random numbers were generated.

Table 4 Results of approximations to π for different RNG's

N	mid-square	Lehmer	Knuth	Marsaglia
1	0.000000	4.000000	4.000000	0.000000
10	2.400000	3.200000	2.800000	2.400000
100	3.240000	3.280000	2.880000	3.040000
1,000	3.896000	3.124000	3.120000	3.136000
10,000	3.989600	3.104400	3.136800	3.160800
100,000	3.998960	3.130680	3.139000	3.149440
1,000,000	3.999896	3.133236	3.141524	3.138980
10,000,000	3.999990	3.134274	3.141964	3.141549
100,000,000	3.999999	3.134406	3.141482	3.141667
1,000,000,000	4.000000	3.134406	3.141579	3.141587

this RNG in the simple method for estimating π . This estimate for π converges toward 4, clearly a bad approximation. The problem is that at some point the pseudorandom numbers generated are all so low that every ball lands inside the circle. As a result, the approximation is very inaccurate.

Although the mid-square RNG is very poor, we introduced it here to demonstrate a number of important points. First, it illustrates the basic procedure of most random number generators: choose a seed, and then use a mathematical algorithm to produce a sequence of pseudorandom numbers. The seed and the mathematical algorithm completely determine the sequence of pseudorandom numbers. Second, it illustrates the idea of periodicity. The period of the RNG in Table 3 is only 4, which is an important reason it should not be used. Every RNG is periodic; many have a period of more than 10^9 . It is important that the period of the RNG be larger than the number of random numbers used in the simulation. Otherwise, the accuracy of the simulation may not increase with the number of pseudorandom numbers generated. Third, it demonstrates the importance of having a RNG with gives

pseudorandom numbers uniformly distributed between 0 and 1. Since the numbers were not uniformly distributed between 0 and 1, the mid-square method gave inaccurate results. Fourth, it shows that even though a RNG gives numbers which appear to be random, it may lead to very inaccurate results.

Linear congruential random number generators

In the late 1940's Lehmer introduced a much better random number generator^(2,3). It was the first *linear congruential generator* (LCG). LCG's are a very common type of random number generator which use integers and modular arithmetic. The seed is an integer, and all of the numbers generated are integers between 0 and some maximum value. The general formula for LCG's is

$$X_i = (aX_{i-1} + c) \text{ mod } m \dots\dots\dots (1)$$

In this equation X_i represents the i^{th} pseudorandom number, with X_0 being the seed which is chosen by the user. There are three constants: a, the multiplier; c, the adder; and m, the modulus, which is chosen to be a very large number. The mod operator simply gives the remainder of dividing by m. For example, $17 \text{ mod } 5 = 2$; $263 \text{ mod } 5 = 3$.

100 = 63; etc. All of the pseudorandom numbers generated are between 0 and m . As a result, the maximum possible period of an LCG is m . (However, if the constants a , c , and m are not chosen very carefully then the period will be much less than m .)

Often it is desired for the random numbers to be between 0 and 1, rather than be integers between 0 and m . In that case one simply divides X_i by m . One can also generate numbers in another range by simple linear transformations. For example, if it is desired that the random numbers be between -1 and +1, then one can divide X_i by m (to get in the range 0 to 1), multiply by 2 (to get in the range 0 to 2), and subtract 1 (to get in the range -1 to +1).

In Lehmer's RNG the constants are $a=23$, $c=0$, and $m=10^8+1$. Column 3 of Table 4 shows the approximations to π with our simple method using Lehmer's RNG. It

performs much better than von Neumann's mid-square method, but it still does not give an accurate result.

There are two problems with Lehmer's RNG. First, the period is relatively small. Since we generated 2×10^9 random numbers and the period cannot be more than 10^8+1 , the sequence must have repeated itself. As a result, increasing the number of balls from 10^8 to 10^9 does not improve the estimate of π .

Second, there are correlations between consecutive pseudorandom numbers. This is demonstrated in Figure 4a, which is an image showing the frequency at which two consecutive numbers are generated. We generated many pairs of numbers between 0 and 1. We used each pair of numbers as cartesian coordinates to determine a pixel in the image, and we added 1 to the pixel value. If the RNG were truly random the

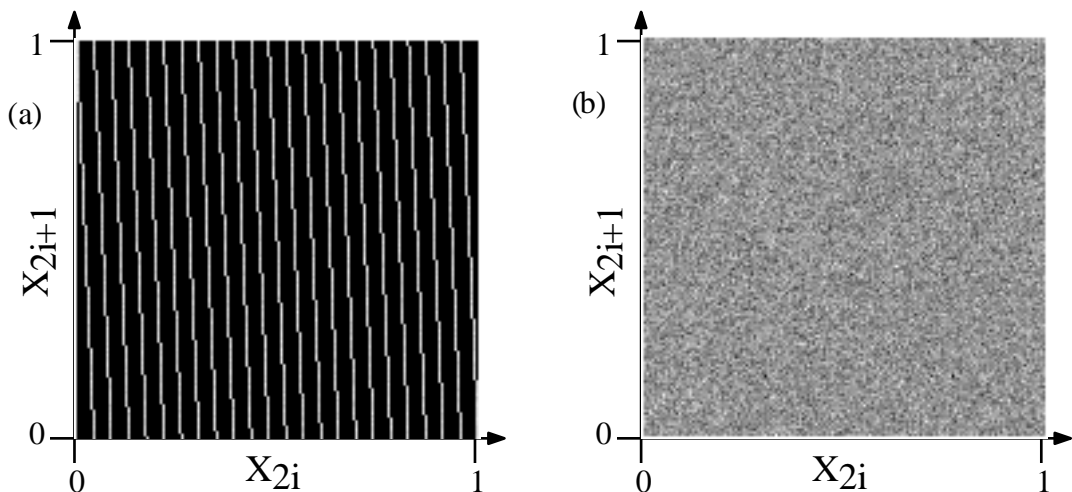


Fig. 4 Images used for testing correlations between pairs of consecutive pseudorandom numbers, for two different LCG's. For each pair of consecutive numbers X_{2i} and X_{2i+1} , the corresponding pixel is located and incremented by 1. (a) Resulting image using Lehmer's LCG, showing high correlation between pairs of consecutive pseudorandom numbers. (b) Resulting image using Knuth's LCG, showing no apparent correlation between pairs of consecutive pseudorandom numbers. There are many other statistical tests that a RNG must pass before it is considered to be a good RNG.

image would look noisy. The lines in the image illustrate that values are correlated.

Subsequent LCG's differ from Lehmer's in the constants a , c , and m . The constants are chosen to give a long period and to reduce the correlation between consecutive pseudorandom numbers. Column 4 of Table 4 shows results of using another LCG (devised by Knuth) where $a=1664525$, $c=0$, and $m=2^{32}$. The estimate of π is much closer to the true value. Figure 4b shows an image similar to Figure 4a but using Knuth's LCG. This shows that consecutive pseudorandom numbers are much less correlated than in Lehmer's LCG. However, with some LCG's correlations show up in higher dimensions than the two dimensions shown in Figure 4. It should be noted that there are many more statistical tests for RNG's.

Modern random number generators

Modern RNG's have much longer periods and perform much better at statistical tests than the common LCG's. One such RNG is due to Marsaglia^(3,5). Its period is longer than 10^{43} . It is based on three RNG's: an LCG and two other types of "simple" random number generators not discussed here. The last column of Table 4 shows the results using Marsaglia's RNG for the estimation of π . For Monte Carlo simulations we highly recommend that a good RNG be used.

Monte Carlo in SPECT

Here we describe brute force Monte Carlo, which is the simplest type of MC. In brute force MC the physics of the imaging process are modelled closely, but it is not very efficient. Many *variance reduction* methods can be used to improve the efficiency of MC programs, but they

are beyond the scope of this paper⁽⁶⁾. For a more sophisticated description of MC in SPECT the reader is referred to⁽⁷⁾.

A Monte Carlo SPECT simulation program takes as input an *activity phantom* and a *tissue map*, and it outputs projection data. The activity phantom corresponds to the radiopharmaceutical uptake distribution in the body, and the tissue map corresponds to the different tissues of the body. (Sometimes an *attenuation phantom* is used rather than a tissue map. The attenuation phantom corresponds to attenuation coefficients of different tissues of the body at some specified energy.) The MC program models photon emission, attenuation and scatter, collimation, and detection. Most of these processes has some random component and require random numbers to be generated. Many photons are simulated—often more than 10^9 —and each photon is followed from its point of emission until it is absorbed in the body, reaches the detector, or passes out of the body undetected.

Let us consider the simulation of one photon. Compared to a ball in the previous simple example, which only landed inside or outside the circle, what can happen to one photon is much more complicated. First, the point of emission is chosen with a random number generator. For a three dimensional phantom, this requires 3 random numbers to be generated. The activity phantom determines the probability of the photon being generated at each point or voxel. In other words, a photon has a higher chance of being emitted from a voxel in the activity phantom which has a high value than a voxel with a low value. As a result, after many photons have been emitted more will have come from regions

in the activity phantom with high activity values than from those which have low values.

After the point of emission is chosen, the direction of the emitted photon is chosen. This direction is chosen with random numbers, since the direction of photon emission is random. For an isotope with one photopeak, the emitted photon will have an energy equal to the photopeak energy. Choosing the point and direction of emission are sufficient to simulate photon emission. For an isotope with multiple photopeaks, the energy of the emitted photon must be chosen. Again a RNG is used to determine the energy of the photon; the probability of the photon having a given energy is determined by the real physical abundance of photons having that photopeak energy.

After emission a real photon may pass through the body without interacting with the body, or it may interact with the body. Photon interaction with a body tissue is another random process, with the probability of any type of interaction given by the cross section corresponding to the tissue, the type of interaction, and the photon energy. As a result, random numbers are used to control the processes of photon interactions with tissue. Random numbers determine how far the photon travels before interacting, what type of interaction takes place, how much energy the photon loses, and the direction that the photon travels after the interaction (if it is not absorbed because of a photoelectric event). If the photon is absorbed, then a new photon is emitted and followed through the body. A photon which interacts and is not absorbed will then continue in a new direction, and it may interact again with the body or it may

travel through the body without interacting. This is how attenuation and scatter in the body are modelled. Note that the MC program can easily keep track of how many times each photon interacts with tissue (i.e. scatters); thus it is possible to obtain scatter information from a MC program. In a real physical experiment, the scatter component can only be estimated.

If the photon passes out of the body without being absorbed, then it will be followed as it propagates through air. It will either reach the collimator or it will miss it and not be detected. This depends on the trajectory of the photon and the location, orientation, and size of the collimator. If the photon reaches the collimator, then it will either pass through the collimator or not. If it passes through the detector and enters the crystal, then it will either be detected or pass through the crystal undetected. Whether it is detected is a random process, depending on the energy of the photon, the attenuation coefficient of the crystal at that photon energy, and the length of the crystal that the photon travels through (i.e., the crystal thickness and the angle that the photon makes with the crystal). Thus photon detection in the crystal is modelled using a RNG. (With some MC programs, the photon can even scatter multiple times in the crystal.)

If the photon misses the collimator, enters the collimator but does not pass completely through, or enters the crystal but passes through undetected, a new photon is generated and the whole procedure is repeated. On the other hand, if the photon is detected then various information about the photon can be recorded. For example, the angle of the collimator, the x- and y-coordinates of the photon, and the photon

energy can be recorded, just as with a real camera. (Energy resolution of the camera can also be modelled with a RNG.) One can use any desired amount of spatial resolution for the camera, and one can use as many energy windows as desired (as long as there is enough computer memory to hold the information). One can also record scatter information, namely whether the photon scattered at least once in the body or passed through without scattering, or even the number of times the photon scattered inside the body.

The whole procedure is repeated for every photon. When the program finishes the user has a projection data set corresponding to a very large number of photons. It is up to the user to decide how many photons to simulate. The number may depend on the desired activity that one wants to simulate, or it may depend on the count level that one desires for the projection data.

The noise level in the projection data depends on the number of photons simulated, as well as on the bin size and the size of the object being simulated. Often one wants projection data with relatively low noise; consequently one has to simulate a very large number of photons. However, the more photons that are simulated the longer the simulation will take.

REFERENCES

1. Tsui BMW, Zhao X, Frey EC and McCartney WH. Quantitative single-photon emission computed tomography: basics and clinical considerations. *Sem Nucl Med* 1994;24:38-65.
2. Knuth DE, *The art of computer programming*, vol. 2, 2d ed. Reading, Mass.: Addison-Wesley, 1981.
3. James F. A review of pseudorandom number generators. *Computer Physics Communications* 1990;60:329-44.
4. Sobol IM. *A primer for the Monte Carlo method*. Boca Raton, Fla.: CRC Press, Inc., 1994.
5. Marsaglia G, Zaman A, and Tsang WW. Toward a universal random number generator. *Statistics & Probability Letters* 1990;9:35-9.
6. Haynor DR, Harrison RL, Lewellen TK, Bice AN, Anson CP, Gillespie SB, Miyaoka RS, Pollard KR, and Zhu JB. Improving the efficiency of emission tomography simulations using variance reduction techniques. *IEEE Trans. Nucl. Sci.* 1990;37:749-753.
7. Ljungberg M and Strand SE. A Monte Carlo program for the simulation of scintillation camera characteristics. *Computer Methods and Programs in Biomedicine* 1989;29:257-72.