Monte Carlo Techniques

The Monte Carlo method (or method of statistical trials) consists of solving various problems of computational mathematics by means of some random process. This process is used for complex modeling problems which are too difficult to solve by simpler modeling techniques which can be handled analytically or via deterministic methods. The Monte Carlo method uses knowledge from past experience to assign probabilities to individual events. A set of rules for a simulation are established, then a series of simulations are performed to determine optimal solutions to the problem or identification of unknown parameters in the system.

The first example that we will consider is a based on the work of H. H. Schmitz and N. K. Kwak (*Operations Research* **20**, 1171-1180 (1970)). They used Monte Carlo methods to determine how many operating rooms were needed, how long the recovery rooms needed to be staffed, and how many beds would be needed in the recovery rooms. They derived their data from plans at Deaconesse Hospital in 1970 to expand their facilities to include 144 new beds. The question arose as to how this would effect the surgery facilities at the hospital. More specifically, how many more surgical procedures will be performed based on this increase in bed capacity and how will this effect the operating and recovery room facilities at the hospital?

The data in 1970 indicate that 42% of the patients staying at the hospital required surgery. This implies that 60 of the 144 new beds would be used primarily for surgery patients assuming that the mix of patients admitted to the hospital came in the same proportions. One could see the possibility that this assumption fails as new facilities would likely encourage more "non-essential" surgical procedures. However, it makes a reasonable first assumption for our modeling purposes. The researchers examined the past history of surgical procedures and concluded that the 60 new beds would result in 3376 new surgical cases giving the hospital a total yearly load of 9669 surgery cases. If these cases are spread evenly over the entire year, then the daily case load would be 27. (If one omitted Sundays and 10 holidays, then this case load would increase to 32 cases per day.)

To answer our questions about details on how many operating rooms are needed, we need more information on the types of procedures performed and the length of time of the operations. Also, we need some idea of how these operations effect the recovery room facilities. With this information, can the hospital determine how it should schedule its surgeries and staff its recovery rooms? An analysis was made on the surgeries performed on 445 patients, and it was found that the length of stay in the operating room is exponentially distributed with an interarrival time mean of 1.03 hours. This indicates that if 4 operating rooms were used then operations would occupy the operating rooms for about 7 hours per day, while 5 operating rooms would cause this to drop to about 6 hours per day. Below we will show how a Monte Carlo simulation can be used to show more about the variation in use of the operating room facilities.

A survey of types of surgery performed gave the results summarized in Table 1. This table shows the length of the surgery, the relative frequency, and the random numbers associated with each type of surgery.

Schmitz and Kwak assume that if the procedure lasts from 0.0-0.5, then they use 0.5 hours. Other procedures are assumed to last the length of time which matches the midpoint of the interval with the last case assumed to last exactly 4 hours. Notice that this assumption does violate the exponential form that they found holds for surgical procedures. A more complicated simulation could be performed by subdividing the random numbers to more closely match the exponential form of the distribution function for time of surgery.

Below we list the rules that are applied in the Monte Carlo simulation.

1. The daily case load is assumed to be fixed at 27 cases.

Type of Surgery	Time Interval	Relative Frequency	Random Numbers
Ear-Nose-Throat	0.0-0.5	15.8	000-157
Urology (To RR)	0.0-0.5	8.4	158-241
Urology (No RR)	0.0-0.5	8.5	242-326
Opthalmology (No RR)	0.0-0.5	5.8	327-384
Other Surgery	0.5-1.0	23.6	385-620
Other Surgery	1.0-1.5	14.6	621-766
Other Surgery	1.5-2.0	9.0	767-856
Other Surgery	2.0-2.5	5.5	857-911
Other Surgery	2.5-3.0	3.4	912-945
Other Surgery	3.0-3.5	2.1	946-966
Other Surgery	3.5-4.0	1.3	967-979
Other Surgery	>4.0	2.0	980-999

2. Random numbers are generated independently for each day.
3. All ENT, urology, and ophthamology cases last 0.5 hours.
4. Half the urology cases and all ophthamology cases do not go to the recovery room.
5. All ENT and the other half of urology cases go to the recovery room and are assumed to stay for 1.5 hours.
6. Any operation over 0.5 hours is considered major and requires 3 hours in the recovery room.
7. Surgery begins at 7:30 a.m.
8. Preparation time is .25 hours in the operating room.
9. It takes .08 hours to transport patients from operating room to the recovery room.
10. It takes .25 hours to prepare the recovery room for the next occupant.
11. Operating rooms are used continuously as need arises with the first one vacated being the first one used.
12. The first vacated recovery bed is the first one filled as needed.
13. If no bed available in the recovery room, then a new one is created.

The attached sheet shows the results of one simulation. Four days were simulated with 5 operating rooms. The attached simulation shows that surgery ended at 14:24, while the recovery room was cleared by 17:44. The latest surgery lasted to 17:30 for the 4 days with the recovery room clearing by 20:36. This was the optimal solution found by Schmitz and Kwak. The use of 4 operating rooms made for days going too long, while 6 operating rooms often completed all surgery before noon.

Problem

MC1. Perform a two simulations each for 4 and 5 operating rooms using a patient load of 32 patients per day (which would be the case load with Sundays and some holidays off). You can obtain your

random numbers from any source that you wish. The computer has several good random number generators which you may wish to use.

Patient #	Random #	Length of Operation	Time Begins	Time Ends	Operating Room #	Recovery Y/N	Recovery Begins	Recovery Ends	RR Bed #	RR Bed Available
1	889	2.25	7.50	9.75	1	Y	9.83	12.83	7	13.08
2	396	0.75	7.50	8.25	2	Y	8.33	11.33	1	11.58
3	358	0.50	7.50	8.00	3	N	-	-	-	-
4	715	1.25	7.50	8.75	4	Y	8.83	11.83	3	12.08
5	502	0.75	7.50	8.25	5	Y	8.33	11.33	2	11.58
6	068	0.50	8.25	8.75	3	Y	8.83	10.33	4	10.58
7	604	0.75	8.50	9.25	2	Y	9.33	12.33	5	12.58
8	270	0.50	8.50	9.00	5	N	-	-	-	-
9	228	0.50	9.00	9.50	4	Y	9.58	11.08	6	11.33
10	782	1.75	9.00	10.75	3	Y	10.83	13.83	4	14.08
11	379	0.50	9.25	9.75	5	N	-	-	-	-
12	093	0.50	9.50	10.00	2	Y	10.08	11.58	5	11.83
13	011	0.50	9.75	10.25	4	Y	10.33	11.83	9	12.08
14	648	1.25	10.00	11.25	1	Y	11.33	14.33	6	14.58
15	527	0.75	10.00	10.75	5	Y	10.83	13.83	10	14.08
16	987	4.15	10.25	14.40	2	Y	14.48	17.48	5	17.73
17	214	0.50	10.50	11.00	4	Y	11.08	12.58	11	12.83
18	474	0.75	11.00	11.75	3	Y	11.83	14.83	2	15.08
19	238	0.50	11.00	11.50	5	Y	11.58	13.08	1	13.33
20	045	0.50	11.25	11.75	4	Y	11.83	13.33	5	13.58
21	408	0.75	11.50	12.25	1	Y	12.33	15.33	3	15.58
22	116	0.50	11.75	12.25	5	Y	12.33	13.83	9	14.08
23	209	0.50	12.00	12.50	3	Y	12.58	14.08	5	14.33
24	048	0.50	12.00	12.50	4	Y	12.58	14.08	12	14.33
25	393	0.75	12.50	13.25	1	Y	13.33	16.33	11	16.58
26	550	0.75	12.50	13.25	5	Y	13.33	16.33	7	16.58
27	306	0.50	12.75	13.25	3	N	-	-	-	-

Modeling a Chain Reaction

Walt Disney studios once filmed a simulated chain reaction[1] which took place as follows. A large number of cocked mousetraps was placed on the floor of a bare room. Each trap was specially built so that when it was sprung it would throw two ping pong balls into the air. Flying ping pong balls that landed on unsprung traps would spring the traps and thereby set more balls flying. This example provides a good model for nuclear reactions (critical mass) or for some epidemics (threshhold phenomenon). The reaction was started by tossing a single ping pong ball into the room. How should the simulation be designed so that the duration of the chain reaction will be reasonable, i.e., the audience must be able to see the process, but it shouldn't last too long? Also, we'd like to determine at the peak of the simulation how many balls are in the air, so that the simulation has the right dramatic effect on the audience. There are three obvious ways to influence the duration of the simulation: Change (1) the flight time of the balls, (2) the number of traps per square foot, or (3) the size of the room (keeping the density of the traps the same). We shall consider each of these events separately.

It can be observed that the flight times of the balls for a given brand of mousetrap are nearly the same. We assume for simplicity that they're identical. After hitting a trap, very few balls are able to rebound enough to hit another trap with sufficient force to spring it. Thus a ball that hits a sprung trap or an unsprung trap becomes dead in most cases. We shall assume that this always happens. A ball that hits the bare floor may or may not rebound enough to be able to set off a trap; it depends on the floor material. At any rate, there is a probability p that a random ball will land on a trap with enough force to spring it (if it is still cocked). The value of p depends only on how far apart the traps are and on the nature of the floor. (The latter is a fourth variable which we can adjust. You should convince yourself that this would have the same effect as changing the spacing of the traps.)

Obviously, the assumptions of identical traps and the balls only being able to spring one trap are weaknesses in the model, but it does allow us to make some pretty good predictions (and in addition there are methods to compensate for these weaknesses in the stochastic model). Our assumption of identical traps allows us to assume that the flight time of a ball works as a unit of time. This would be the hardest parameter to adjust in our simulation of a chain reaction. Also, if we wanted to increase the simulation time by this parameter, we would lose some of the synchrony from distributed flight times. Let n be the length of time from the start of the simulation until b_n balls are in the air. Assume that b_n is much less than the total number of balls. A first ball is thrown into the room and either 0 or 2 balls are released. The expected value for this first generation is $b_1 = 2p$ balls. At the second, generation the expected value is $b_2 = (2p)^2$. Clearly, at the n^{th} generation, $b_n = (2p)^n$ is the expected number of balls in the air. This implies that $n = \ln(b_n)/\ln(2p)$. This expression is valid for small times where only a small percentage of the traps have been sprung.

One interesting calculation is that if we want to have the same percentage of balls in the air for two rooms of differing size, then it requires increasing the original room b_n times to increase the time by $\ln(b_n)/\ln(2p)$ units. This means that at time n, if b_n/B balls are in the air for the first room, then for the same percentage of balls to be in the air for a room b_n times larger it requires a time $2n$. The calculation is as follows. Suppose it takes k units of time for x balls to be in the air in the first room which is $100x/B$ percent of the balls. For the second room $100x/B = 100b_nx/b_nB$, which implies b_nx balls are in the air. But $b_nx = (2p)^n(2p)^k$, which yields $n + k = \ln(b_n)/\ln(2p) + \ln(x)/\ln(2p)$ or an increase of $\ln(b_n)/\ln(2p)$ time units.

[1]This example is taken from E. Bender's book *Introduction to Mathematical Modeling.*

The above computations are only for the initial times when few traps have sprung. To analyze the intermediate times another type of analysis is needed. Let N be the number of balls in flight at the time $t = n$ and U be the number of unsprung traps out of a total of M. The conditional probability of having exactly $2B$ balls in flight at time $n+1$, given T traps are hit, is

$$P(B|\mathrm{T}) = \binom{\mathrm{T}}{B}\left(\frac{\mathrm{U}}{M}\right)^{B}\left(1-\frac{\mathrm{U}}{M}\right)^{\mathrm{T}-B},$$

where $\binom{\mathrm{T}}{B} = \frac{\mathrm{T}!}{B!(\mathrm{T}-B)!}$. This expression is a direct result of the fact that if N balls hit T traps, then U/M is the probability that an unsprung trap is hit, while $\left(1-\frac{\mathrm{U}}{M}\right)$ is the probability that the ball hits a trap that has already been hit. To release $2B$ balls then B unsprung traps must be hit. This is a binomial distribution. If we assume that no trap is hit by more than one of the N balls (which would be valid for $N \ll M$), then we can use a binomial distribution to see that the probability of T traps being hit satisfies

$$H(\mathrm{T}) = \binom{N}{\mathrm{T}} p^{\mathrm{T}}(1-p)^{N-\mathrm{T}}.$$

These probabilities can be used to show that what is likely to happen at the next generation, but analysis is difficult for determining how actual simulations might proceed to completion.

We can use our Monte Carlo simulation method to simulate how a complete chainreaction might occur. Below we present a MATLAB program which allows one to view how the chainreaction proceeds for a case where there a 100 mousetraps. The user inputs a probability p. The computer generates a 10×10 matrix of 1's indicating no unsprung traps. The program proceeds by determining the probability that a trap is hit based on the number of balls in the air using a random number generator.

MATLAB Program for Monte Carlo Simulation

```
%User input for probability of hitting a trap.;
p = input('Probability of a hit:  p = ');
%Initialize the number of balls in the air, b, the generation time, i,;
%and the matrix and vector of unsprung traps, mt and xmt, respectively;
b=1;
i=0;
mt=ones(10,10)
xmt=mt(:);
%Compute the total number of unsprung traps.;
u=sum(xmt);
fprintf(' Unsprung Traps = %3.0f, Balls in Air = %2.0f\n',u,b)
pause  %Strike any key to continue.
%Loop for continuing the process until no balls are in the air.;
while b > 0
%Increase the generation time.;
i=i+1;
%Use a random number generator to determine how many traps are hit.;
th1=(rand(1,b)<=p*ones(1,b));
th2=th1(:);
th=sum(th2);
%Use the random number generator to determine exactly which traps are;
%hit between 1 and 100.
c1=ceil(100*rand(1,th));
c2=c1(:);
%Compute how many balls are released based on unsprung traps and;
%transform those traps to ones which are sprung.;
b=2*sum(xmt(c2));
xmt(c2)=0*xmt(c2);
mt=reshape(xmt,10,10)
u=sum(xmt);
fprintf(' Unsprung Traps = %3.0f, Balls in Air = %2.0f\n',u,b)
pause  %Strike any key to continue.
end
fprintf(' Generations = %2.0f\n',i)
```

A modified version of this program was run 100 times with each probability, p, listed in the table below. Below we present tables which summarize some of the statistical results of the simulations along with the run which has the peak number of balls in the air and the longest simulation of each set.

$p = .80$	Mean	Median	σ^2	Peak	Long
u	51.9	35	31.6	25	32
b_{max}	17.2	20	11.4	40	12
i	11.3	14	6.67	12	22

$p = .90$	Mean	Median	σ^2	Peak	Long
u	34.0	23	28.6	13	31
b_{max}	28.6	32	13.4	50	20
i	11.5	13	4.68	11	22

$p = .95$	Mean	Median	σ^2	Peak	Long
u	20.8	19	12.8	16	33
b_{max}	37.0	38	8.49	62	26
i	12.5	12	2.33	10	18

$p = .99$	Mean	Median	σ^2	Peak	Long
u	15.6	14	9.71	9	14
b_{max}	42.4	42	9.14	72	30
i	12.0	12	1.78	11	16

From the tables we see that the higher probabilities have a more dramatic rise with more balls in the air, while the lower probabilities have slightly longer duration on average (especially considering the higher failure rate to even start). The variation between simulations becomes much more evident as the probability of a hit drops though again this is slightly skewed because of the number of early failures. The simulations do not show how the duration would change if the number of mousetraps were increased though this would not be difficult to add to the program.

The Monte Carlo simulations are easy to implement; however, they can become very computationally intensive when there are a large number of events to be considered (say as one would find in a nuclear reaction in trying to determine critical mass). We would like to develop a more deterministic scheme which would better describe the situation for very large numbers of traps and balls. For large unsprung traps, U, and balls in the air, N, the binomial distributions of P (probability of $2B$ balls in flight given T traps hit) and H (probability of T traps hit) can be approximated by normal distributions. The means for P and H are UT/M and pN, respectively. The variances of P and H are given by $\mathrm{UT}(M-\mathrm{T})/M^2$ and $Np(1-p)$, respectively. If N_n is the expected average number of balls in the air at time n, then

$$\begin{aligned} N_{n+1} &= 2pN_n\mathrm{U}_n/M, \\ \mathrm{U}_{n+1} &= \mathrm{U}_n - \tfrac{1}{2}N_{n+1}. \end{aligned}$$

From these formulae, we have a recursive formula for the average number of balls in the air and the number of unsprung traps. Like most recursive formulae, it gives you a local answer but fails to show the more general behavior of the scheme.

Let $f(n)$ be the fraction of unsprung traps at time n. From the recursive formulae above

$$\frac{\mathrm{U}_{n+1}}{M} = \frac{\mathrm{U}_n}{M} - \frac{N_{n+1}}{2M} \quad \text{or} \quad f(n+1) - f(n) = -\frac{N_{n+1}}{2M.}$$

But

$$N_{n+1} = \frac{2pN_n\mathrm{U}_n}{M} = 2pN_nf(n) \quad \text{and} \quad \frac{N_n}{2M} = f(n-1) - f(n),$$

so

$$f(n) - f(n+1) = 2p[f(n-1) - f(n)]f(n). \tag{1}$$

This is a nonlinear second order difference equation. This is a deterministic model for the average fraction of unsprung traps. The initial conditions for this difference equation are $f(0) = 1$ and $f(1) = 1 - p/M$. With this information, (1) can be solved numerically by a very simple program. The results of a series of simulations which were run until the number of balls in the air dropped below $2f(1)$ with $M = 100$ are presented in the table below.

p	u	b_{max}	i
.80	31.0	20.1	15
.90	20.3	29.4	13
.95	16.2	35.4	12
.99	13.5	39.5	11

The values in this table compare very favorably to the ones computed using the Monte Carlo simulation and the computer program and simulation time is substantially less. This type of simulation fails to show how much variation might be expected from different runs. There are no techniques that we have developed to analyze nonlinear difference equations. Nonlinear difference equations as we've seen before can exhibit quite complicated behavior.

In Bender's book, he takes this example one step further to show that the difference equation can be transformed into a differential equation. To do this we write a Taylor's series expansion of degree two about $f(n)$. This gives

$$f(n+1) = f(n) + f'(n) + \tfrac{1}{2}f''(n)$$

and

$$f(n-1) = f(n) - f'(n) + \tfrac{1}{2}f''(n).$$

Substituting these expressions into our difference equation, we find

$$f(n) - [f(n) + f'(n) + \tfrac{1}{2}f''(n)] = 2p[f(n) - f'(n) + \tfrac{1}{2}f''(n) - f(n)]f(n).$$

This simplifies to

$$f''(n) = \frac{4pf(n) - 2}{2pf(n) + 1}f'(n) = \left(2 - \frac{4}{2pf(n) + 1}\right)f'(n),$$

with initial conditions $f(0) = 1$ and $f'(0) = 1 - \frac{p}{M}$, the expected fraction of traps that will spring initially. Upon integratation we find

$$f'(n) = 2f(n) - \frac{2}{p}\ln[2pf(n) + 1] + C,$$

where the constant $C = \frac{2}{p}\ln[2p+1] - 2 - \frac{p}{M}$. This equation has no analytical solution. The best way to see its behavior is to use some numerical scheme.

The numerical simulations for $f(n)$ are given for several values of p, using a Runge-Kutta scheme.

p	u	b_{max}	i
.80	30.6	20.3	15
.90	19.8	29.7	13
.95	15.6	33.7	12
.99	12.7	37.4	11

Notice that the solution to this differential equation agrees reasonably well with the solution of the difference equation though it is even lower than the mean of the Monte Carlo simulations.