AN INTRODUCTION TO
MONTE CARLO METHODS

Monte Carlo is a fancy term for simply using random values. Monte Carlo describes any technique utilizing random values although more complex methods within usually bear equally complex names since, to paraphrase Dr. Ulfarsson's cynicism on the matter, researchers like fancy terms for simple things and complex terms for everything else. The methods presented in this introduction provide a practical foundation upon which even complex research questions may be successfully attempted.

Contents

Introduction

Before covering Monte Carlo Integration or Monte Carlo Probability or Monte Carlo Projections, it must be recognized that Monte Carlo *Anything* relies upon randomly generated values. When Monte Carlo was developed, this was done mechanically via flipping coins, rolling dice, or spinning a roulette wheel. This was as labor intensive as it was time consuming; it wasn't until the advent of powerful computers and the creation of convoluted algorithms to formulate pseudo random values, that this process became practical.

The pseudo random generator creates values forming a uniform distribution, meaning all values between the bounds have equal probabilities of selection. Other distributions such as the Normal, Gumbel, etc. may be created from U(a,b), but those transformation are noted here solely to stress the potential and flexibility of the distribution. To acquire a value between 0 and 1 on a 32-bit system, the computer may:

1. Take an initial value x-$_0$, called the seed, either inputted manually or acquired from the computer's internal clock.
2. Calculate x_n as 7^5 times x_{n-1} modulus $(2^{31}-1)$, where n is an element of the natural numbers.
3. Put it between 0 and 1 by dividing x_n by $(2^{31}-2)$.
4. Repeat 2-4 until the desired number of random values are generated.

(Mooney 12-13)

In Matlab®, the function `rand` provides a pseudo random value between 0 and 1.

Probability

Flip a coin. Tally heads. Flip again. Tally heads. Flip again. Tally tails…

From a child's diversion to a prisoner of war's distraction, nearly everyone has tried flipping a coin over and over to see how often it lands heads or tails. Depending on the flipper's patience (or boredom), many flips will be made, and, as the phenomena known as the Law of Large Numbers dictates, the proportion heads and proportion tails will approach their true proportions of exactly .5.

That was a Monte Carlo simulation for probability. Granted time replaced the coin with a computer's pseudo random, it does not preclude these examples from *family game night*.

Example 1.1

What is the probability of getting a seven or eleven when rolling two dice in craps?

Write code in a Matlab® M-file:

```
%This solves craps probability
% via a Monte Carlo simulation

n = 1000000 %samples
SoE = 0; %Seven or Eleven

for i=1:n
    switch (roll(6)+roll(6));
        case 7
            SoE = SoE + 1;
        case 11
            SoE = SoE + 1;
    end
end

seven_or_eleven = SoE/n
```

Execute and compare the result with the actual probability of .222. Try it a few times for different sample sizes, n, and see how greatly the solutions vary.

Although the exact proportion can easily be found by creating a square addition table from one through six, tallying the times seven and eleven appear and dividing by the possibilities, such simple methods are not always apparent (or existent).

Example 1.2

In Risk®, the attacker rolls three red dice, and the defender rolls two white dice. The highest red die is compared to the highest white die. The next highest red die is compared to the lowest white die. For each comparison, the defender loses an army if the red die is greater than the white die; otherwise, the attacker looses an army. Each roll, how many armies are expected to be lost by the attacker? the defender?

Write code in a Matlab® M-file:

```
%This solves Risk® probability
% via a Monte Carlo simulation

n = 1000000 %samples
dl = 0; %Defend loss
for i=1:n
    x = sort([roll(6) roll(6)
          roll(6)],'descend');
    y = sort([roll(6) roll(6)]
          ,'descend');
    if (x(1) > y(1))
        dl = dl + 1;
    end
    if (x(2) > y(2))
        dl = dl + 1;
    end
end
attack_loss_per_roll = 1-dl/n
defend_loss_per_roll = dl/n
```

Simulation provides 0.92 and 1.08 armies.

Integration

Monte Carlo integration utilizes The Mean Value Theorem for Integrals, which states, as recalled from calculus, that, if the function f(x) is continuous on the interval upon which integration is to take place, there is a constant average value f_{ave} such that:

$$f_{ave} = \frac{1}{b-a}\int_a^b f(x)\,dx \qquad \text{(Equation 1.1)}$$

Rearrangement of Equation 1.1 provides:

$$\int_a^b f(x)\,dx = f_{ave}(b-a) \qquad \text{(Equation 1.2)}$$

Thus Equation 1.2 shows that the integral is the average value of the function multiplied by the interval. The interval is known; to get the average value use a Monte Carlo simulation.

Monte Carlo simulation is much like implementing a survey, except here the population sampled is not people but rather is the function between the lower and upper bounds of integration. Likewise, many statistical assumptions hold true. For example, just as a person randomly selected from the population is assumed to provide a fixed, non-random response to the survey, so too does the function of a randomly selected value have a fixed response. From the responses, the average may be found by taking an arithmetic mean.

Putting everything together creates the expression:

$$\int_a^b f(x)\,dx = (b-a)\left(\frac{\sum^{n} f(r_n)}{n}\right) \qquad \text{(Equation 1.3)}$$

where r is a random number between a and b (r_n is used to emphasize that the random number is not a constant – a new random is selected for each term though it may be the same as one prior by virtue of luck). The sample size, n, is chosen for a desired accuracy.

Recall from statistics that a larger n reduces the variance, thus increasing the confidence of the result. Unlike a real survey of a population, a simulation on a computer does not incur significant costs for an increase in sample size, so ignore the variance and relax as the computer runs the algorithm below for as many millions of iterations as time allows.

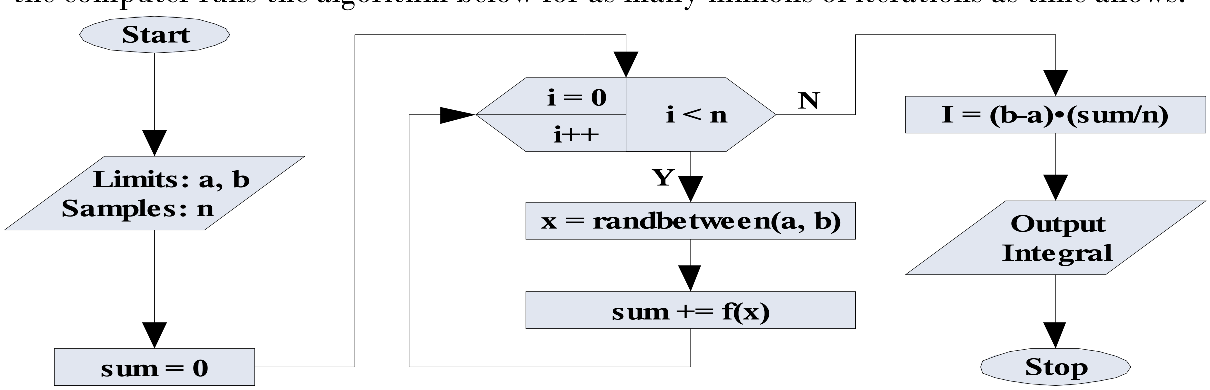


Figure 2.1. A computing algorithm for Monte Carlo integration

Example 2.1

Solve: $F(x) = \int_0^{10} x \, dx$

Write code in a Matlab® M-file:

```
%This code solves y=x via a
% Monte Carlo simulation

a = 0;  %lower bound
b = 10; %upper bound
n = 1000000; %samples

sum = 0;

for i=1:n
    x = randbetween(a, b);
    fOFx = x;
    sum = fOFx + sum;
end

Integral = (b—a)*(sum/n)
```

Execute and compare the result with the analytically acquired solution of 50. Try it a few times for different sample sizes, n, and see how greatly the solutions vary.

Example 2.2

Solve: $F(x) = \int_0^{10} x^{(2-x)}+1 \, dx$

Write code in a Matlab® M-file:

```
%This code solves y=x^(2-x)+1
% via a Monte Carlo simulation

a = 0;  %lower bound
b = 10; %upper bound
n = 1000000; %samples

sum = 0;

for i=1:n
    x = randbetween(a, b);
    fOFx = x^(2-x)+1;
    sum = fOFx + sum;
end

Integral = (b—a)*(sum/n)
```

This function is rather complicated to integrate analytically, but Monte Carlo integration quickly provides an approximate solution of 12.40.

Calculus students may inquire why Monte Carlo integration is used instead of Riemann Sums, Trapezoidal Rule, etc. The answer is bias. Consider y = x as shown in Figure 2.2, and see that Riemann Sums will always be above the actual value — smaller step-sizes cause improvement, but it will always be high. Monte Carlo integration is unbiased. As Figure 2.3 shows, if done with Monte Carlo integration a hundred-thousand times with sample size of ten-thousand, the distribution of error is centered about zero.

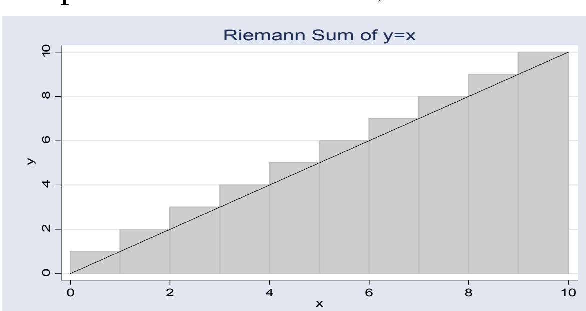


Figure 2.2. Riemann Sum of y = x

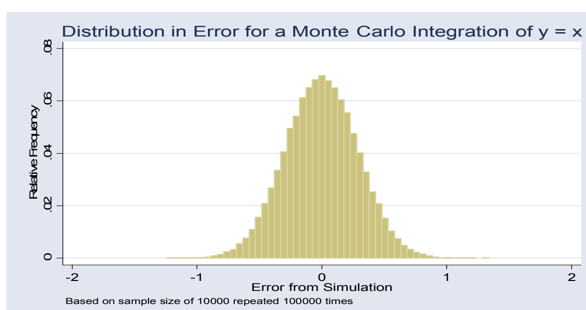


Figure 2.3. Error Distribution of Monte-Carlo

If Riemann Sums for midpoint were used above, the result would be correct; unfortunately knowing which method and step-size to use requires understanding the function. Even for small step sizes, of, say, a millionth, if the function also had a period of a millionth there could be a large systematic error. With Monte Carlo, understanding may determine if a smaller sample size works; but simply adding a few million to n works too.

Projections

Monte Carlo simulations may be employed in estimating a project's net cost, completion time, etc. Begin by breaking a project into steps for which to estimate a lower bound and upper bound for the variable of interest. Then the simulation draws random values between the bounds for each step and sums across all steps. The process is repeated thousands of times and a cumulative frequency "S-Curve" is drawn from the results. The S-Curve allows one to see the most a project will, say, cost at some percentage of the time.

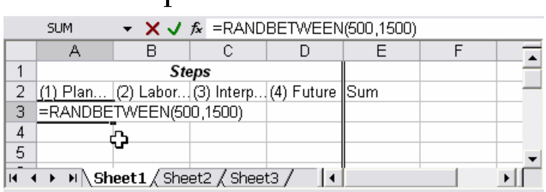
Forecast simulations are typically done with spreadsheet software. In Excel®, the formula for a random number between two bounds is =RANDBETWEEN(lower,upper). This function represents a uniform distribution; but, other distributions may be used.

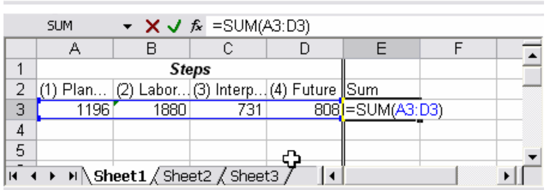
Example 3.1

Given the table below showing a four step project with estimated upper and lower bounds for the cost, forecast the amount of money to budget to be 85% confident that the budget is adequate.

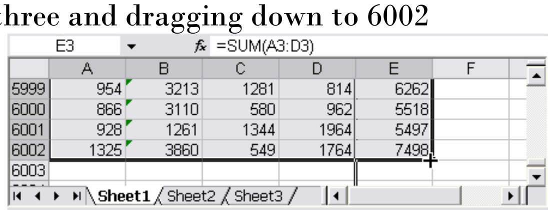
Step	*Lower*	*Upper*
(1) Planning	$500	$1500
(2) Laboratory time	$1000	$5000
(3) Interpretation	$500	$1500
(4) Future planning	$500	$2000

1st: Create a spreadsheet

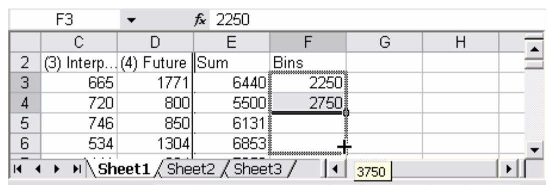




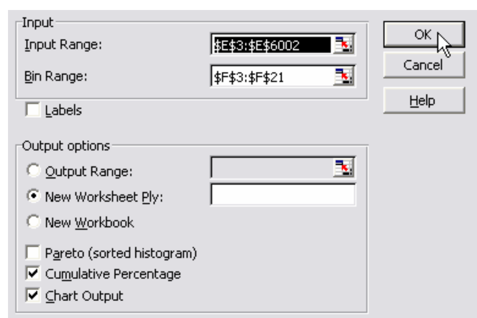
2nd: Iterate 6000 times by selecting row three and dragging down to 6002



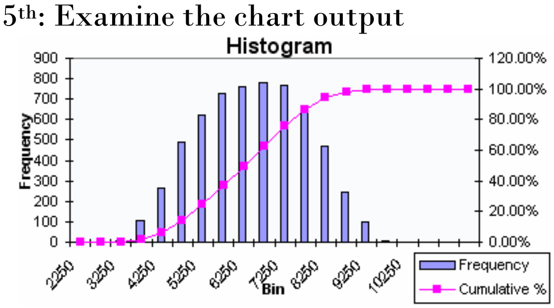
3rd: Create histrogram bins. Enter 2250, 2750 and drag down to 11250



4th: Create histogram.
Tools >> Data Analysis >> Histogram



5th: Examine the chart output



Trace 85% to the Cumulative % line and trace down to the bin. So budget away approximately $7750 to be 85% sure.

Functions

There are many common, however cryptic, tasks employed in implementing a Monte Carlo simulation. Using functions not only makes code more readable but also makes coding easier. Try, for example, a dice roll or draw from a bounded uniform distribution.

Example 4.1

Create a function to simulate the roll of a die with sides number of sides.

Write code in a Matlab® M-file:

```
function x = roll(sides)
%This function behaves like a
% dice roll for a die of given
% number of sides.

x = int32(sides*(rand)-.5)+1;
```

Note: Matlab®'s int32 function rounds to the nearest integer; thus, subtracting by .5 is required.

Example 4.2

Create a function to simulate draws from a uniform distribution with bounds [a,b].

Write code in a Matlab® M-file:

```
function x = randbetween(a,b)
%This function returns a
% pseudo random value between
% bounds a and b

x = ((b-a)*(rand)+a);
```

Simply put, a random number from 0 to 1 is multiplied by the interval b-a and a is added to put it in place.

Conclusion

Integration, probability and projections were covered in such a way as to provide a basic, practicable understanding of Monte Carlo methods, but it must be stressed that this has been merely an introduction. Although nothing further is necessary to implement these methods, more is required to do so effectively. Study of advanced topics, including the use of distributions other than the Uniform distribution (Pareto, Exponential, Normal, etc.), is recommended for simulation of non-Uniform processes and for improvement of simulation efficiency.

References

Al-Dahhan, Muthanna. (2004). *Introduction to Computing and Computer Applications*. St. Louis, MO: Washington University Chemical Engineering Department.

Milton, Susan J., & Arnold, Jesse C. (2003). *Introduction to Probability and Statistics* (4th ed.). New York: McGraw-Hill Higher Education.

Mooney, C. Z. (1997). *Monte Carlo Simulation* (Sage University Paper on Quantitative Applications in the Social Sciences, series no. 07-116). Thousand Oaks, CA: Sage.

Razgaitis, Richard. (2003). *Dealmaking Using Real Options and Monte Carlo Analysis*. Hoboken, NJ: John Wiley & Sons.

Robert, Christian P., & Casella, George. (2004). *Monte Carlo Statistical Methods* (2nd ed.). New York: Springer Science+Business Media.

Special thanks to Dr. Al-Dahhan for his advice over the project and to Prof. Nobs for his practical perspective on simulations.