

"...using Monte Carlo methods and simulated annealing in a multi-modal hill-climbing approach toward an optimal configuration in molecular-dynamic stochastic processes, with something quantum thrown in for good measure. "

-too many buzzwords

Computationally inclined scientists love buzzwords nearly as much as businessmen do. I frequently encountered references to the "Monte Carlo method" in my studies and perusings across the sciences, always unexplained, a buzzword for something probabilistic and computational. Without knowing exactly what it was, I read about this magic method's application to knot theory, protein folding, particle dynamics, integration and optimization. In absence of convenient definition, the seed of research was planted. Its fruit is this text.

The Monte Carlo method sprang from the head of Stanislaw Ulam, a mathematician working with John von Neumann (noted computer scientist and all-round good guy) on the Manhattan Project. In Eckhardt (1987), Ulam describes the genesis of the idea:

The first thoughts and attempts I made to practice [the Monte Carlo Method] were suggested by a question which occurred to me in 1946 as I was convalescing from an illness and playing solitaires. The question was what are the chances that a Canfield solitaire laid out with 52 cards will come out successfully? After spending a lot of time trying to estimate them by pure combinatorial calculations, I wondered whether a more practical method than "abstract thinking" might not be to lay it out say one hundred times and simply observe and count the number of successful plays. This was already possible to envisage with the beginning of the new era of fast computers, and I immediately thought of problems of neutron diffusion and other questions of mathematical physics, and more generally how to change processes described by certain differential equations into an equivalent form interpretable as a succession of random operations. Later ... [in 1946, I] described the idea to John von Neumann, and we began to plan actual calculations.

Monte Carlo is the great casino-city of Monaco, famous for its games of chance. The method's name is thus inspired, as the roots of its power lie in probability and the atom bomb (the greatest gamble of all), its origins in card-counting and idle hands.

At its heart, the Monte Carlo method is a means to avoid very hard math. It dodges arduous symbolic manipulations, agonizingly complex systems and bogglingly infinite possibilities by wandering around the problem's domain, poking it, trying a few random examples to get a general idea (and a good estimate) of what's going on.

How it Works: *Expected Values and Estimations Thereof*

In a "probability experiment", one is concerned with the possible outcomes of an event, and their frequency of occurrence. Each event in the set of all outcomes has a probability of occurring. Because something has to happen in the end, the sum of all probabilities must be 100%. (Even nothing is something. If it's possible that nothing might happen, then nothing has a probability of happening, and is thus an event.) This relationship between events and their likelihood is the probability distribution for the experiment. A coin flip is the classic example: The three events resulting from the flip (heads-up, tails-up, or edge-up) have respective probabilities of 49.999999%, 49.999999%, and not very likely.

When probability-scientists get bored with plain events, they assign random variables to the outcomes. This associates a quantifiable value with each event, that ideally has some significance to the subject of the experiment. (In our coin example, we might assign a random variable representing the money (in dollars) that we'd win or lose by betting on the coin's fall, say heads= 0.25, tails= -0.25, edge=1,000. By contemplating various properties of the distribution, we'd be able to make a statement about our probable earnings in this betting-game

The expected value of a random variable is the mean of its value across all possible events, weighted against their probability of happening. The expected value of our coin-betting random variable would represent our average winnings over an infinite number of tosses. In this case, the expected value is

$$E = (0.49999999 \cdot 0.25) + 0.49999999 \cdot -0.25 + (0.00000002 \cdot 1000) = 0.00002$$

which is not very good money (but none the less better than a loss). Although we can't wait around until infinity is over to reap our earnings, we can expect that the average of our winnings will approach the expected value (namely nothing) as the number of tosses increases. Once we land a coin on its edge, we have an extra \$1000 in our pocket. But the average winnings over the fifty million tosses it took us to do so will be very near to two thousandths of one cent. Thus by sampling a largish number of experiment-outcomes, we can obtain a pretty good estimate of the random variable's expected value.

The intuition is clearer when dealing with discrete events, but the fun and cleverness of probability works equally well with continuous probability distributions. In such cases, the event is associated with a real value, and the probability distribution becomes a probability density function. This function maps the real value (the event) between a and b (the endpoints of the range of possible-event values) to a probability (between 0 and 1). Over the domain of possible event-values, the density function's definite integral must be 1. This is the same as saying that the sum of all probabilities, over all possible events, must be 100%.

The expected value of a random function $f(x)$ with a probability distribution of $P(x)$ is

$$E(f) = \int_a^b P(x)f(x)dx$$

$E(f)$ may not be an easy thing to know in advance. If either the random variable or the probability distribution is particularly gnarly, it could be a pain to integrate. Thankfully, it turns out that $E(f)$ can be very nicely estimated by taking N samples $x_1, x_2, x_3 \dots x_n$ from x (according to $P(x)$), and calculating the average of $f(x)$.

$$E(f) \approx \frac{1}{N} \sum_{n=1}^N f(x_n)$$

As N gets larger, the quality of the estimate improves. The upshot is that with a good bit of simple, old-fashioned number crunching, we can avoid a nasty integral and get fairly close to the expected value we were looking for.

The utility doesn't stop with finding expected values. This method of sampling-and-averaging (the Monte Carlo method) allows us to estimate the value of any integral, even the nastiest! We can shoehorn any definite integral that we want (the integral of $g(x)$, from a to b , we'll call it I) into looking like an expected value for a continuous random variable:

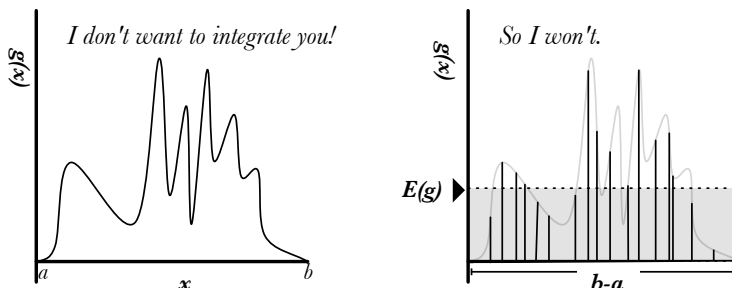
$$I = \int_a^b g(x)dx \quad P(x) = \begin{cases} \frac{1}{b-a}, & a \leq x \leq b \\ 0, & \text{otherwise} \end{cases} \quad I = (b-a) \int_a^b P(x)g(x)dx$$

$P(x)$ above is the uniform distribution, wherein all possible events are equally likely. It's the least exciting probability density function in the world, but (as we see here) very useful. The factor $(b-a)$ takes the "average" expected value and stretches it over the integral's range, counterbalancing the effect of $P(x)$.

By picking N samples uniformly from the range (a, b) , we can achieve an estimate for the integral:

$$I = (b-a)E(g)$$

$$I \approx (b-a) \frac{1}{N} \sum_{n=1}^N g(x_n)$$



The area under the curve is equal to the area under the expected value.

Monte Carlo integration works. It's as simple as its intuition suggests. One of its great benefits is that the estimation is dimension-independent, free of the *curse of dimensionality* that plagues so many other techniques, wherein computational effort increases as the dimension of the domain increases. An estimate by the Monte Carlo method produces the same quality of results, with efficiency dependent only upon the size of the sample set and not the complexity of the function. While easier problems may not benefit from the time spent gathering and calculating thousands of sample points, complex functions stand to gain quite a bit, time-and-effort-wise, by settling for the pretty-good Monte Carlo estimation.

Application

An intuitive demonstration of the Monte Carlo method (in the form of a Java applet) has been developed by your humble author. It allows the creation of arbitrary two-dimensional figures upon a canvas, whose areas are subsequently estimated by interactively sampling points upon that canvas. Conceptually, the function being integrated is a two-dimensional function,

$$f(x,y) \rightarrow \{0,1\}, 1 \text{ where } (x,y) \text{ is within the drawn figure, and } 0 \text{ where it is not.}$$

Where the mathematics above may be unwieldy to some, the basic idea is most approachable when visually presented. The applet and its source code are available at

<http://cs.oberlin.edu/~dadamson/montecarlo/> (see also the **Appendix** for code and documentation)

Finding area (under a curve, or in a funny blue shape) is fine and dandy, but Monte Carlo is so useful because many problems from any number of fields can be fit into an integration-mold without too much trouble. Computational physicists and molecular chemists, for example, use Monte Carlo to estimate the multi-dimensional integrals that so often arise in their fields.

A popular extension of the Monte Carlo method is the *Markov Chain* Monte Carlo method, or MCMC. A Markov Chain is a sequence of "states" where each state is dependent only on the state before it, and here serves as the probability distribution, the sample-factory. Instead of randomly sampling points in a statistically independent, uncorrelated way (as described in the last section), a sequence of samples is generated by a "random walk" through the problem space, "collecting" the value at each location as it moves (according to the rules of the Markov chain) through the space, choosing its next point based upon its contribution to the integral. MCMC allows for complex, state-dependent probability distributions, such as those related to the motion or interactions between particles.

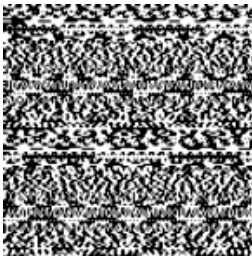
Pseudo-Randomness

Monte Carlo is a probabilistic method, it relies on its samples to be uniformly distributed across the sample space. Monte Carlo is also a computational method, in that we rely on computers to do MC's

boring, repetitive, mechanical number-crunching for us. However, computers are not probabilistic. Instead, they are very definitely *deterministic*, in that what they do (and the numbers they produce) must follow a programmable pattern, something not-at-all random. Yet we expect their aid in creating random samples? Agony! Horror! How can we use computers to implement the Monte Carlo method, without a random number generator!??

Option one is to reach beyond the machine and into the world, to draw upon the randomness in nature. The chip-maker Intel uses the fluctuation in the temperature of its processors as a source of randomness, telephone companies and network-software manufacturers have drawn upon the length of and time between connections. While these measures follow no pattern, they still incur the curse of clumping, of irregular and poorly distributed samples. A sudden spike in processor use will skew Intel's generator, and a slow day on the switchboard will do likewise to AT&T's. Such "truly random" sources are both unpredictable and unrepeatable - contradictory to the heart of randomness as it may be, it is useful to repeat the same experiment more than once, to demonstrate identical results.

Option two: We fake it. Arthur C. Clarke observed that a sufficiently advanced technology is indistinguishable from magic, and we shall twist that to observe that a sufficiently convoluted algorithm is indistinguishable from randomness. The built-in "pseudo-random" number generators that see use in spreadsheets and video games and other off-the-shelf applications are not very random: The distance (or period) between repeated sequences is far too short. Poor random number generation results in skewed data - certain points will be hit more than others, leading to "clumps" in the data that leave some regions under-sampled. The pseudo-random number generator provided with the Java programming environment is dramatically un-random. The figure below was generated by "random" coin flips using Java. Notice the thick bands and the repeated shapes within the image. The blotchy square shown will repeat itself identically after four iterations - we have no control over this repetition, it's very undesirable.



Designing something that is repeatable and that will not create patterns is very hard, and countless researchers have been active in such pursuit. "Good" randomness techniques do exist today, with interesting names like "Mersenne Twister" and "Mother-of-all". The research and implementation of these methods will await this author's future research.

Conclusion

The Monte Carlo method is a simple concept, intuitive for the visual thinker and sensible to the symbol-worker. Its subtleties are many, its applications range widely. With MC, the computer is a tool, neither an end in itself nor a demanding master, making a job easier instead of complicating a simple task. May it be so more often.

Bibliography

- Eckhardt, Roger. “Stan Ulam, John von Neumann, and the Monte Carlo method,” **Los Alamos Science**, Special Issue (15), 131-137. 1987.
- “Markov Chain Monte Carlo,” **Wikipedia** (electronic encyclopedia). 2005.
Online at http://en.wikipedia.org/wiki/Markov_Chain_Monte_Carlo
- Metropolis, Nicholas and Stanislaw Ulam. “The Monte Carlo method,” **Journal of the American Statistical Association**, 44 (247), 335-341. 1949.
- “Monte Carlo Chapter,” **Computational Science Education Project** (electronic book). 1995.
Online at <http://csep1.phy.ornl.gov/CSEP/MC/MC.html>
- “Monte Carlo Method,” **Risk Glossary**. Contingency Analysis, 2003.
Online at http://www.riskglossary.com/articles/monte_carlo_method.htm
- Python, M. **Monte Carlo and the Quest for the Holy Estimator**. *facetious bibliographic entry*.
- Unner, Eric. “Generating Random Numbers,” **Embedded Systems Programming**. 2004.
Online at <http://www.embedded.com/showArticle.jhtml?articleID=20900500>