Monte Carlo Methods Introduction

I don't particularly want to get into politics. Some classes should be apolitical simply as a respite for all the politicized classes (so that you can go back into those classes refreshed and ready to argue again). But in the case of Monte Carlo methods, the best example by far that I can think of is political. I will at least remove it from the present context somewhat by making it a political example from 2016. Here we go....

Suppose you have poll data from each of 50 states. Each poll has some uncertainty. For example polls in the State of Arizona might show Clinton beating Trump with a 52% to 48% margin. There is some uncertainty in this poll though, so ***a poll with a 4% margin for Clinton doesn't mean that 100% of the time Clinton will get Arizona's 11 electoral college votes.*** With a 4% margin, and some polling uncertainty, we might assume that Clinton will beat Trump in Arizona 80% of the time and get those 11 electoral college votes. However, it might go the other way 20% of the time and in that case Trump gets those 11 electoral college votes. I just made those numbers up. Obviously the actual probabilities depend on the margin and your assessment of the uncertainty in the polling data.

Now repeat this imaginary analysis for each of the other 49 states. Each of those states will have its own poll results, and from the margin in that state's poll results, and your estimate of the uncertainty of the polling in that state, you will get some percentage chance that Clinton gets that state's electoral college votes.

Now here is the problem. There are 2^{50} possible outcomes, right?! Every stinking state could go one way or the other. If there were only three states in the US, you would have $2^3 = 8$ possible outcomes, and each of those outcomes would have a probability, and each of those outcomes has an electoral college tally. It's trivial to assign the probabilities and have the computer do all 8 possibilities and then compute what fraction of the time Clinton gets a majority of the electoral college.

But there are 50 states, and each specific outcome has a ridiculously small probability, and every one of these outcomes results in a different electoral college tally. 2^{50} is about 10^{15}. Even if your computer can do 1,000,000 tallies a second, it would take about 10^9 seconds ***which is about 30 years*** to come up with an answer to the question, "what is the chance that Clinton will get a majority of the electoral college votes?" The intractability of this problem is one reason why there is so much focus on swing states. One way to make the problem tractable, is to assume that 40 or so of the 50 states are not in play. Then you run the computer model on the remaining 2^{10} possibilities, and 2^{10} is 1024, and a computer can easily do 1024 possibilities just as easily as it can do 8, and then tabulate the fraction of times that Clinton gets an electoral college majority from within those 1024 possibilities.

Remember that ***an integral can always be done — in principle — by chopping up the interval into a***

large number of parts and computing the height of each part, taking the product of the width and height of each part to get an area, and then summing up the large number of little areas. But from the above example, you get a hint that ***doing summations numerically — while always possible in principle — might somehow fail in practice due to the sheer amount of time required to exhaustively do the summations.*** I could explain a bit more, by going back into how integrals are related to sums, but I am hopeful that you already see the problem.

If you aren't happy with the political example, or you think it might be bizarrely non-representative of most real-world problems, as a second example, consider ray tracing. Each pixel on a digital frame of an animated movie has to be given an intensity and a color and that has to be done for every one of the roughly 4,000,000 pixels per frame of every one of the 150,000 or so frames in a 1 1/2 to 2 hour movie, which is 600,000,000,000. It's bad, but it if a few thousand computers in a render farm can do 100,000,000 pixels a minute, they can render the whole movie in 100 hours.

But this isn't the whole problem. You get very unrealistic-looking animation if you don't get much more realistic. Every pixel is actually a recursive ray tracing problem. Let me describe ray tracing. A pixel might be a pixel showing an animated character's iris, and the animation artist has built a model of the scene in which the iris has some intrinsic color and some reflectivity. The animation artist has also placed lighting into the scene and each light source has its own color and intensity.

The ray tracing program can't just assign a color and intensity to the pixel based on the intrinsic color and reflectivity of the iris, because what the iris looks like depends on what light sources are illuminating it! This is not too bad a calculation yet if there are a finite number of nearby light sources. ***However***, if you want to make the scene look even more realistic, you have to account for the fact that there may be nearby objects to the character's face, like a brightly lit and gauzy curtain that the figure is standing near, that also contribute to how much their iris is lit up. The curtain is not one of the primary light sources, but it is lit up by the primary light sources, and even worse, if it is a gauzy curtain some of the light that comes from its direction comes from the scene behind it. ***So now the problem has become a recursive problem: to figure out how the iris is lit up, we have to figure out how much each portion of the nearby curtain is lit up,*** and we might even have to have a model of the scene behind the curtain.

This is a computational quagmire. To figure out the color and intensity of a pixel representing a portion of a character's iris is we now have to figure out what light is coming from all the enormous number of bits of cloth that together make the gauzy curtain that is illuminating the person. We could call those rays secondary because they illuminate the thing that is going to make the primary ray going that goes to the vantage point that the computer is modeling.

An approach to dealing with the computational cost is to randomly sample a small number of those bits of cloth, and figure out how much each of those is illuminated. ***We would need that random sample to be representative of the whole curtain in order to achieve realism.***

Anyway, getting any degree of realism is obviously computationally horrendous. It is in principle doable by computer, but in practice even the huge render farm of thousands of computers at a company like Pixar cannot do the calculation by brute force. Cutting down the problem is going to somehow involve cutting down the secondary and tertiary rays to some randomly selected subset of all the possible rays, and those rays need to be representative. So now we have two examples showing that the problem of doing sums or integrals by computer seems to only work for simple situations, and for more complex situations we need representative samples.

This problem of recursion was encountered during the design of the atomic bomb. Each fission reaction is caused by other fission reactions. A fission reaction releases releases a lot of energy, but equally important it typically releases 2 to 3 neutrons, each of which can travel through the fissile material and trigger another fission reaction which in turn releases 2 to 3 neutrons that then can each trigger another reaction. This is called a "chain reaction." Now the issue is that whether or not each neutron causes another reaction depends in a detailed way on (a) the neutron's speed (it turns out that slow neutrons work better, somewhat surprisingly), and (b) how close to the edge of the fissile material the neutron is, because it might just escape the core of the bomb into the surrounding non-fissile material. You can see that a chain reaction is another recursive problem that you have no hope of exhaustively evaluating.

So, I have been telling you guys that Mathematica can do almost any integral and that it is your friend for calculating areas under curves if you either (a) haven't taken Calculus BC, or (b) you run into an integral of a function that they didn't cover in Calculus BC, but now we see that there are lots of integrals and sums that cannot be done in practice.

In my next handout, we are going to learn the Monte Carlo method by playing with it.