16 MONTE CARLO SIMULATION

In Chapters 14 and 15, we looked at different ways of using randomness in computations. Many of the examples we presented fall into the class of computation known as **Monte Carlo simulation**. Monte Carlo simulation is a technique used to approximate the probability of an event by running the same simulation multiple times and averaging the results.

Stanislaw Ulam and Nicholas Metropolis coined the term Monte Carlo simulation in 1949 in homage to the games of chance played in the casino in the Principality of Monaco. Ulam, who is best known for designing the hydrogen bomb with Edward Teller, described the invention of the model as follows:

> *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,[109] 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.[110]*

The technique was used during the Manhattan Project to predict what would happen during a nuclear fission reaction, but did not really take off until the 1950s, when computers became both more common and more powerful.

---

[109] Ulam was probably referring to the ENIAC, which performed about 10^3 additions a second (and weighed 25 tons). Today's computers perform about 10^9 additions a second.

[110] Eckhardt, Roger (1987). "Stan Ulam, John von Neumann, and the Monte Carlo method," *Los Alamos Science*, Special Issue (15), 131-137.

Ulam was not the first mathematician to think about using the tools of probability to understand a game of chance. The history of probability is intimately connected to the history of gambling. It is the existence of uncertainty that makes gambling possible. And the existence of gambling provoked the development of much of the mathematics needed to reason about uncertainty. Contributions to the foundations of probability theory by Cardano, Pascal, Fermat, Bernoulli, de Moivre, and Laplace were all motivated by a desire to better understand (and perhaps profit from) games of chance.

16.1 Pascal's Problem

Most of the early work on probability theory revolved around games using dice.[111] Reputedly, Pascal's interest in the field that came to be known as probability theory began when a friend asked him whether or not it would be profitable to bet that within twenty-four rolls of a pair of dice he would roll a double 6. This was considered a hard problem in the mid-17th century. Pascal and Fermat, two pretty smart guys, exchanged a number of letters about how to resolve the issue, but it now seems like an easy question to answer:

- On the first roll the probability of rolling a 6 on each die is 1/6, so the probability of rolling a 6 with both dice is 1/36.
- Therefore, the probability of not rolling a double 6 on the first roll is 1 - 1/36 = 35/36.
- Therefore the probability of not rolling a double 6 twenty-four consecutive times is $(35/36)^{24}$, nearly 0.51, and therefore the probability of rolling a double 6 is 1 - $(35/36)^{24}$, about 0.49. In the long run it would not be profitable to bet on rolling a double 6 within twenty-four rolls.

Just to be safe, let's write a little program, Figure 16.1, to simulate Pascal's friend's game and confirm that we get the same answer as Pascal. When run the first time, the call `checkPascal(1000000)` printed

```
Probability of winning = 0.490761
```

This is indeed quite close to 1 - $(35/36)^{24}$; typing `1-(35.0/36.0)**24` into the Python shell produces `0.49140387613090342`.

[111] Archeological excavations suggest that dice are the human race's oldest gambling implement. The oldest known "modern" six-sided die dates to about 600 BCE, but Egyptian tombs dating to about 2000 BCE contain artifacts resembling dice. Typically, these early dice were made from animal bones; in gambling circles people still use the phrase "rolling the bones."


```
def rollDie():
    return random.choice([1,2,3,4,5,6])

def checkPascal(numTrials):
    """Assumes numTrials an int > 0
       Prints an estimate of the probability of winning"""
    numWins = 0
    for i in range(numTrials):
        for j in range(24):
            d1 = rollDie()
            d2 = rollDie()
            if d1 == 6 and d2 == 6:
                numWins += 1
                break
    print('Probability of winning =', numWins/numTrials)
```

Figure 16.1 Checking Pascal's analysis

16.2 Pass or Don't Pass?

Not all questions about games of chance are so easily answered. In the game craps, the shooter (the person who rolls the dice) chooses between making a "pass line" or a "don't pass line" bet.

- Pass Line: Shooter wins if the first roll is a "natural" (7 or 11) and loses if it is "craps" (2, 3, or 12). If some other number is rolled, that number becomes the "point" and the shooter keeps rolling. If the shooter rolls the point before rolling a 7, the shooter wins. Otherwise the shooter loses.
- Don't Pass Line: Shooter loses if the first roll is 7 or 11, wins if it is 2 or 3, and ties (a "push" in gambling jargon) if it is 12. If some other number is rolled, that number becomes the point and shooter keeps rolling. If the shooter rolls a 7 before rolling the point, the shooter wins. Otherwise the shooter loses.

Is one of these a better bet than the other? Is either a good bet? It is possible to analytically derive the answer to these questions, but it seems easier (at least to us) to write a program that simulates a craps game, and see what happens. Figure 16.2 contains the heart of such a simulation.