Week 13: Monday, Apr 25

Logistics

- HW 7 is extended to the morning of 4/27 (in class or by CMS)
- Project 3 is due Friday, May 6.

Quick review of probability

Monte Carlo methods involve computations done with the help of random numbers. In order to reason about Monte Carlo methods, we need a little background in probability theory. I assume that you've seen some probability theory before, but let's quickly review some basics.

When we do an experiment, there are a variety of possible outcomes that could result. An *event* is a set of possible outcomes that we might care about. Mathematically, we might describe the experiment by the random variable X. The probability that the outcome of the experiment is in an event A is $P\{X \in A\}$. The probability maps events (sets) to numbers in $[0, 1]$; there are some axioms, which I will leave you to look up in a more comperhensive text. For the moment, let's just concentrate on probabilities for the simplest cases. For a discrete random variable, we write

$$P\{X \in A\} = \sum_{x \in A} p_X(x)$$

where p_X is a *probability mass function* (pmf) which is everywhere between zero and one and which sums to one when the sum is taken over all possible outcomes. Similarly, for a continuous random variable, we write

$$P\{X \in A\} = \int_A f_X(x)\, dx$$

where $f_X(x)$ is a *probability density function* (pdf). When the outcomes are integers or real numbers, we sometimes also care about the *cumulative distribution function* (cdf)

$$F_X(x) = P\{X \leq x\}$$

which we can get by summing the mass function or integrating the density function. The cdf is a monotonically increasing functions with limiting values $\lim_{x\to-\infty} F_X(x) = 0$ and $\lim_{x\to\infty} F_X(x) = 1$.

The *expected value* of a function g of a random variable X is

$$E[X] = \int_\Omega g(x) f_X(x)\, dx;$$

in the discrete case, the integral is replaced by a sum. The variance of X is

$$\mathrm{Var}[X] = E[(X - E[X])^2] = E[X^2] - E[X]^2.$$

The standard deviation is the square root of the variance, and we can think of it as a measure of how far, on average, X is from its expected value.

Random variables X and Y are *independent* if for general choices of events A and B we have $P(\{X \in A\} \cap \{Y \in B\}) = P\{X \in A\} \cdot P\{Y \in B\}$. In simple Monte Carlo calculations, we typically run repeated experiments that are *independent and identically distributed* (i.i.d.). If $X_1, X_2, \ldots, X_N$ are independently drawn from the same distribution, then the sample mean

$$\bar{X} = \frac{1}{N} \sum_{j=1}^{N} X_j$$

is a random variable that is approximately normal (Gaussian) with mean $E[X]$ and variance $\mathrm{Var}[X]/\sqrt{N}$.

The Monte Carlo idea

Monte Carlo methods use random numbers to compute something that is not random. In the abstract, we write some quantity of interest A as

$$A = E_f[V(X)],$$

where X is a collection of random variables whose joint distribution is f (sometimes written $X \sim f$) and $V(x)$ is some quantity determined by X. A Monte Carlo code generates many samples X_k, $k = 1, \ldots, N$, from the distribution f, and then computes the approximate answer

$$A \approx \hat{A}_N = \frac{1}{N} \sum_{k=1}^{N} V(X_k).$$

If the samples X_k are independent, the error is roughly $\sigma/\sqrt{N}$, where $\sigma^2 = \mathrm{var}_f(V(X))$ is the variance of the random variable $V(X)$. If we don't know the variance of $V(X)$ analytically (which is typically the case), we can use the estimate

$$\hat{\sigma}_N^2 = \frac{1}{N-1} \sum_{k=1}^{N} (V(X_k) - A)^2.$$

Sometimes we're sloppy and divide by N; if N is small enough that this makes a significant data, we ideally should run more experiments! When we approximate A by $\hat{A}_N$, we call $\hat{\sigma}_N$ an "error bar", since it describes a measure of the statistical error in our problem (the radius of a symmetric 67% confidence interval). The error bars are not the same as error *bounds*, of course, but they are useful for reasoning about the order of magnitude of the errors we expect to see.

Because statistical error is $O(1/\sqrt{N})$, it tends to be very expensive to get high accuracy with Monte Carlo methods. For some problems, though, particularly those in high dimensions, Monte Carlo methods are the most practical choice. The basic idea of Monte Carlo is simple, if expensive; much of the cleverness in Monte Carlo methods goes into *variance reduction*, which at least reduces the constant in the $O(1/\sqrt{N})$ expression. The good side of statistical error is that it is usually at least possible to estimate its order of magnitude (via error bars).

Examples

Monte Carlo methods have relatively low accuracy compared to deterministic methods, but they are particularly useful in a few cases:

1. Some problems are naturally probabilistic, and a Monte Carlo method may be an almost-direct translation of the problem statement. If we don't mind low accuracy, this can be a very effective way to get a feel for the answer before diving into a more exact calculation (which we might have to spend more time debugging). The standard advice is to only use Monte Carlo for things that cannot be well managed by deterministic methods; this sort of exploratory computation might be an exception.

2. The cost of deterministic methods often grows *exponentially* with the dimension of the ambient space. This causes a problem when we're

interested in even moderately high dimensions. For computing integrals in high-dimensional spaces (including the sort of position-and-direction coordinates we need to describe particles in scattering problems like the one in HW 3), a Monte Carlo method is often appropriate.

3. Sometimes we are driven by data, and the data that we have is too huge to process all at once. Sampling the data by Monte Carlo methods can be a very effective approach in this case for the same reason it is effective for high-dimensional problems: the cost depends on the number of samples we draw, and not on the size or dimension of the underlying thing from which our samples are drawn.

Random number generation

In order to run Monte Carlo simulations, we need a source of pseudo-random numbers. One could teach an entire class on how to produce pseudo-random number generators, but we will simply state that it is a tricky business and you should use a well-designed library routine for your day-to-day draws of random bits or of numbers that are uniformly distributed in the interval $[0, 1]$. In MATLAB, you can use `rand` to get such uniformly distributed random samples (and `randn` to get samples from a standard normal distribution). For our purposes, we simply need to know how to turn such uniform sampling procedures into methods to sample from other distributions. We will take up a few strategies for this in the next lecture.