Title

Author

Given the choice between an electrical pump and a hand pump for getting water out of a well, most people would choose the electric. Yet the hand pump is quite reliable and can be used when other sources of power are not available. Similarly, Monte Carlo methods are remarkably versatile algorithms for solving difficult numerical problems when other methods are not practical. We focus in this case study on three uses of Monte Carlo methods: for function minimization, for discrete optimization, and for counting.

Function Minimization

A convex function $f(\mathbf{x})$ has a unique minimum that can be found using a variety of methods, including Newton's method and its variants, conjugate gradients, and (if derivatives are not available) pattern search algorithms. For functions that are nonconvex, such as that in Figure 1, these algorithms will find a **local minimizer** such as x_1 but are not guaranteed to find the **global minimizer** x^*.

Figure 1. *A standard minimization algorithm might find the minimum marked by 'x' if we start at '*', but Monte Carlo minimization seeks the rightmost minimum. (This will be redrawn.)*

Minimization Using Monte Carlo Techniques

Monte Carlo methods provide a good means for generating starting points for optimization problems that are nonconvex. In its simplest form, a Monte Carlo method generates a random sample of points in the domain of the function. We use our favorite minimization algorithm starting from each of these points, and among the minimizers found, we report the best one. By increasing the number of Monte Carlo points, we increase the probability that we will find the global minimizer.

This method can be improved somewhat by using extra information about the function $f(\mathbf{x})$ that we are minimizing. For example, suppose we know a **Lipschitz constant** L for our function, so that for all $\mathbf{x}$ and $\mathbf{y}$ in the domain,

$$|f(\mathbf{x}) - f(\mathbf{y})| \leq L\|\mathbf{x} - \mathbf{y}\|.$$

To make the example specific, suppose that $L = 1$ and the domain is just the real line. If we know that $f(1) = 2$ and $f(4) = 0$, then using the Lipschitz inequality with $x = 1$, we obtain $|f(1) - f(y)| <= (1)||1 - y||$, or $|2 - f(y)| <= |1 - y|$.

POINTER.
For background on probability, consult a standard text such as [**?**].
Simulated annealing is discussed in more detail by Kirkpatrick, Gelatt, and Vecchi [**?**].
Heath [**?**, Chapter 13] and Knuth [**?**, Chapter3] give information on the generation of (pseudo-)random samples. See also http://random.mat.sbg.ac.at/literature/ and http://random.mat.sbg.ac.at/links/rando.html
For testing random number generators, consult [**?**, Chapter3] or http://stat.fsu.edu/g̃eo/diehard.html
More information on random counting algorithms and the estimation of partition functions can be found in papers by Kenyon, Randall, and Sinclair [**?**], Beichl and Sullivan [**?**], Caflisch [**?**], and Beichl, O'Leary, and Sullivan [**?**].

Therefore, letting y range between -1 and 3, we see that $f(y)$ is bounded below by 0. Therefore, our global minimizer cannot lie here. This saves us the work of generating more points in this interval.

In the next challenge, we experiment with Monte Carlo minimization.

CHALLENGE 1. Consider the function `myf.m` (on the website), with domain $0 \leq x \leq 7$.
(a) Write a function to generate 500 uniformly distributed starting points on the interval $[0, 7]$ and use your favorite minimizer (perhaps `fmincon`), starting from each one, to find a local minimum. Make a graph that shows which starting points result in which local minimizer. (There are many ways to do this; just make sure that your graph displays the information clearly.)
(b) $L = 18.1$ is a Lipschitz constant for the function `myf.m`. Try to speed up the minimization using this information. Document the changes to your function and compare the performance of the two methods.
(c) (Extra) Try Monte Carlo minimization on your favorite function of n variables for $n > 1$.