Daniel Votaw

Computational Physics

Final Project

This project demonstrates *simulated annealing* (SA) as an optimization technique. SA is a stochastic method for locating global extrema of functions. The stochasticity is used to guarantee that the algorithm will eventually converge to the **global** extremum, whereas deterministic optimizers face a general problem that they can get stuck in local extrema. If you have some function which potentially has many local extrema, but you need to find the global extremum, deterministic methods are not suitable. For purposes of this project, I’ll focus on cases where the goal is to find a global minimum, but the algorithm can be modified in the obvious way to look for maxima instead (to maximize f(x), just minimize –f(x)).

The function to be minimized will be referred to as the “cost function” f, and it takes some D-dimensional vector **x** as an input. The possible values of the components of **x** live in some predefined parameter space Ω. Most of the time, we are not as interested in the minimum value of the cost function, but rather the values of its inputs such that it is minimized. Therefore mathematically, the algorithm is designed to find:

The idea behind simulated annealing is basically an extension of the Metropolis algorithm (discussed in class), but where the temperature of the Boltzmann acceptance distribution is “cooled” as a function of Monte Carlo time in order to drive the system to its “ground state” (i.e., find the minimum of the cost function). The ability of the Metropolis accept/reject routine to accept a move to a point with a higher cost, which may seem counterintuitive, is what allows SA to evade getting trapped in local minima, and what will eventually guarantee its convergence to the true global minimum once the parameters and “cooling schedule” have been set adequately. As is often the case with Monte Carlo methods, we will see that this algorithm is often sub-optimal, or “overkill”, in low-dimensional spaces, but extremely efficient in high-dimensional spaces.

We will see that for a given set of algorithm parameters, at low dimensions, SA will often iterate over the entire parameter space multiple times, making it more efficient to simply use a brute force search. However for the same set of algorithm parameters, at high dimensions, the SA algorithm may only need to iterate over a very small fraction of the space in order to find the global minimum. For a particular implementation shown in this project, the dimensionality where SA begins to outperform brute force in terms of volume is somewhere between 4 and 5. And the dimension-dependence of the volume performance is very strong; in this particular case, the scaling seems to obey a power law of the form (Fraction of space iterated over) ∝ Dp, where p ≈ -10.7 (see volume\_analysis.plt). Of course SA is a stochastic method, so the exact values of the runtime and fraction of the space iterated over vary with each run. They will depend on the initial conditions, the shape of the cost function, and the algorithm parameters, but to analyze the “average behavior”, we can simply take multiple runs and average the results.