Chapter 10

Optimization

In this chapter we consider a very different kind of problem. Until now our prototypical problem is to compute the expected value of some random variable. We now consider minimization problems. For example we might have a purely non-random problem: find the mininum of some function $H(x)$ where x ranges over X, and find the minimizer. Or we might want to minimize some function which is the mean of some random variable.

10.1 Simulated annealing

We consider the problem of minimizing $H(x)$, a non-random problem a priori. We will look at simulated annealing which is especially useful in situations where $H(x)$ has local minima which cause many of the standard minimization methods like steepest descent to get “stuck.” Explain the name.

The basic idea is to study the probability measure

$$\frac{1}{Z} e^{-\beta H(x)} \tag{10.1}$$

on X and let $\beta \to \infty$. In this limit this probability measure should be concentrated on the minimizing x or x’s. If we just set β to a large value and pick an initial state at random, then the algorithm can get stuck near a local minimum near this initial value. The key idea behind simulated annealing is to start with a small value of β and then slowly increase β as we run the MCMC algorithm. In physics, β is propotional to the inverse of the temperature. So letting β go to infinity means the temperature goes to zero. So this is often called “cooling.”

One thing we need to specify for the algorithm is how fast we increase β. A standard choice is to let $\beta = \rho^n$ where n is the time in the MC simulation and ρ is a parameter that is just slightly bigger than 1. There is no obvious way to choose ρ. One should try different values and diferent choices of the initial condition. One can then compare the final state for all the MCMC runs and take the one with the smallest H.

So the algorithm looks like this:

1. Pick an initial state X_0, an inital $\beta = \beta_0$, and a cooling rate ρ.

2. For $i = 1, \cdots, N$, let $\beta_i = \rho\beta_{i+1}$ and use some MCMC algorithm with β_i to generate the next state X_i.

3. The final state X_N is our approximation to the minimizer.

We could also compute $H(X_i)$ at each time step and keep track of which X_i is best so far. It is possible that along the way we get a state X_i which is better than the final X_N. If it is expensive to compute $H()$ this may not be worth doing.

Example: We start with a trivial example that illustrates how the method avoids getting stuck in local minima.

$$H(x) = sin^2(\pi x) + \alpha x^2 \tag{10.2}$$

where $\alpha > 0$ is a parameter. Obviously the minimum is at $x = 0$, and there are local minima at the integers. If α is small some of these local minima are very close to the true minima. In our simulations we take $\alpha = 0.1$. With this value of α the local minima near 0 are pretty close to the global minima.

We use Metropolis-Hastings with a proposal distribution that is just uniform on $[X_n - \epsilon, X_n + \epsilon]$. We take $\epsilon = 0.2$ and $X_0 = 10.$. We start with $\beta_0 = 0.1$ and raise β by the rule $\beta_n = \rho\beta_{n-1}$ where we will try several different ρ. We consider three different choices of ρ. For each choice we run four simulations. The simulations are run until β reaches 100. Note that this correpsonds to very different numbers of time steps. We use a logarithmic scale on the horizontal axis in the plots. Since $\log(\beta) = n\log(\rho)$ this is a linear scale for the number of Monte Carlo steps.

In the first figure, figure 10.1, we use $\rho = 1.005$. The cooling is too rapid in these simulations and so the chain gets stuck in local minima that are not the global minima.

In the second figure, figure 10.2, we run four simulations with $\rho = 1.0001$. This does better than the previous simulation. One of the four simulations finds the global min while the other three get stuck in an adjacent min.

In the third figure, figure 10.3, we take with $\rho = 1.0000001$. Three of the four runs find the global min, but one still gets stuck in a local min.

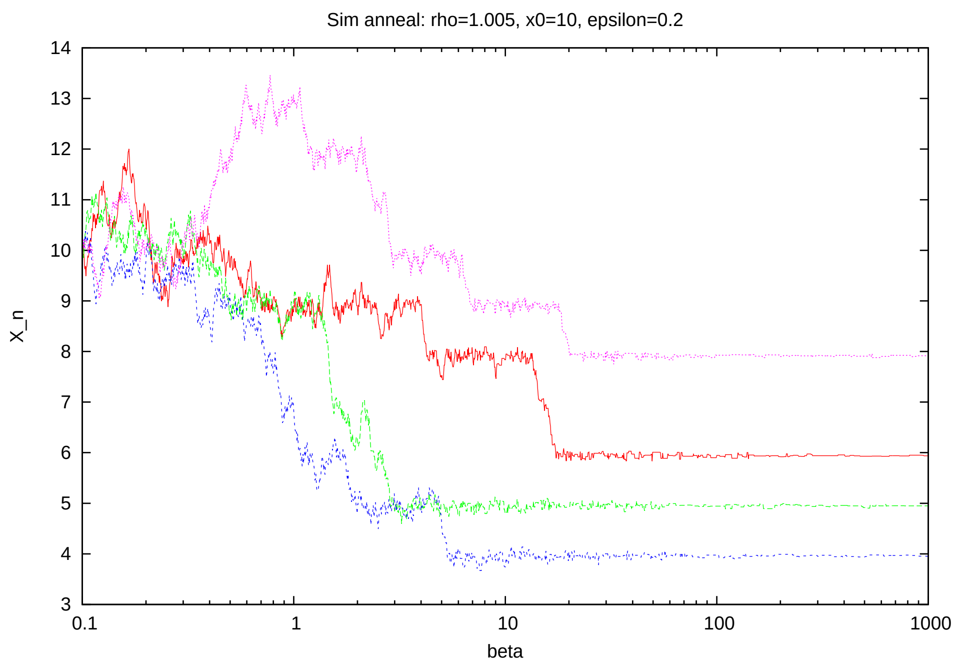

Figure 10.1: Simulated annealing $\rho = 1.005$.

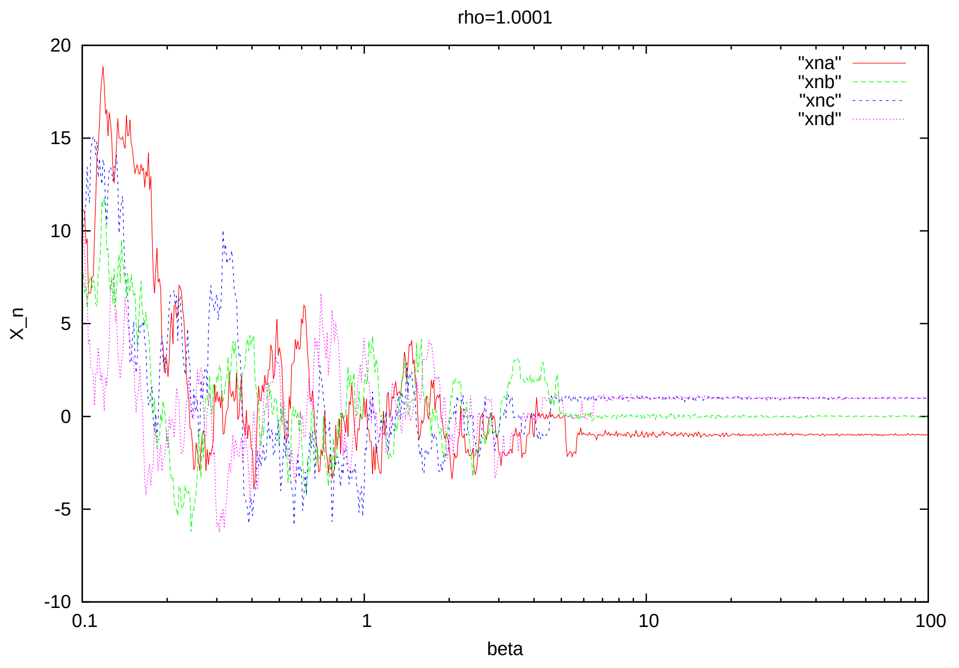

Figure 10.2: Simulated annealing $\rho = 1.0001$.

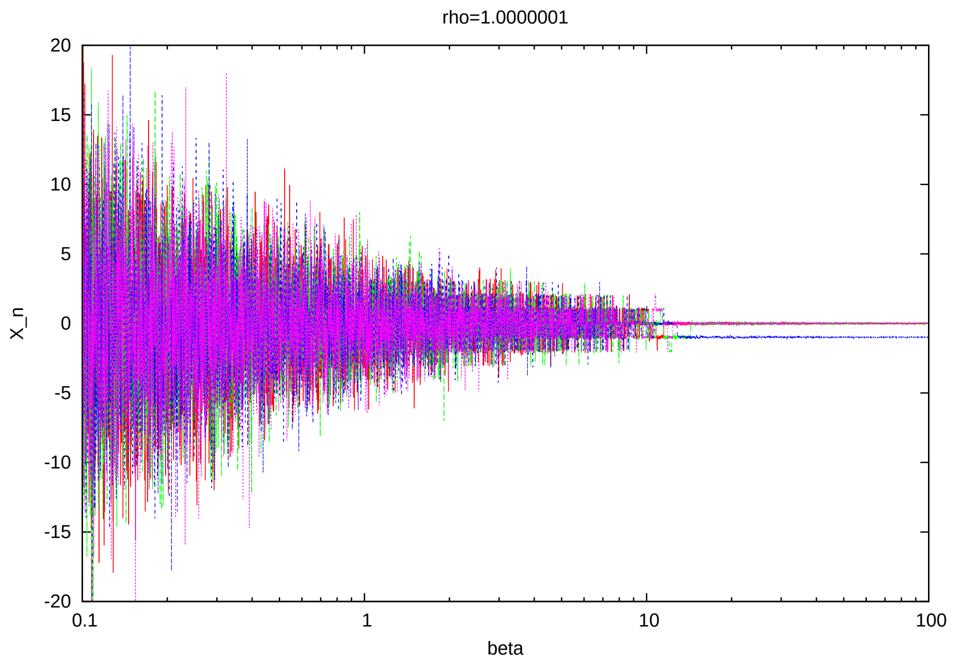

Figure 10.3: Simulated annealing $\rho = 1.0000001$.

Example: Simulated annealing can be used for discrete problems as well. Here is a famous example - the travelling salesperson problem. We have n cities labelled $1, 2, \cdots, n$. The "cost" of travelling from i to j is $c(i, j)$. We assume this is symmetric. The salesperson must start in some city and make a circuit that visits every city exactly once, ending back in the starting city. We call this a tour. So a tour is a permuation π of $1, 2, \cdots, n$. We start in $\pi(1)$, go to $\pi(2)$, then to $\pi(3)$, and so on to $\pi(n)$ and finally back to $\pi(1)$. The cost of the tour is

$$H(\pi) = \sum_{i=1}^{n-1} c(\pi(i), \pi(i+1)) + c(\pi(n), \pi(1)) \tag{10.3}$$

We want to find the tour that minimizes this cost.

We can use simulated annealing with the Metropolis Hastings algorithm. We need a proposal distribution. Here is one. Let π be the current tour. We pick two distinct cities i, j uniformly at random. We interchange these two cities and reverse the original tour between i and j. Do example.

Example: We are designing integrated circuit chips. We have a very large number of "circuits". There are too many to get them all onto one chip. We have to use two chips. Certain circuits need to be connected to other circuits. These connections are "expensive" is the two circuits are on different chips. So we want to decide how to put the circuits on the two chips to minimize the number of connections that must be made between the two chips. Let m be the number of circuits. Let a_{ij} equal 1 if there has to be a connection between circuits i and j, and equal 0 if no connection is needed. It is convenient to encode the board that circuit i is on with a variable that takes on the values -1 and 1. Call the boards A and B. We let $x_i = 1$ if circuit i is on board A and $x_i = -1$ if circuit i is on board B. Note that $|x_i - x_j|$ is 1 if the two circuits are on different boards and $|x_i - x_j|$ is 0 if the two circuits are on the same board. So the total number of connections needed between the two boards is

$$\frac{1}{2} \sum_{i,j} a_{i,j} |x_i - x_j| \tag{10.4}$$

(We set $a_{ii} = 0$.) We want to choose x_i to minimize this quantity. This problem has a trivail solution - put all the circuits on the same board. We need to incorporate the constraint that this is not allowed. Note that $|\sum_i x_i|$ gives the difference in the number of circuits on the two boards. We don't need this to be exactly zero, but it should not be two large. One model would be to add a constraint that this quantity is at most ?? Another approach is to add a penalty term to the function we want to minimize:

$$\frac{1}{2} \sum_{i,j} a_{i,j} |x_i - x_j| + \alpha [\sum_{i=1}^{m} x_i]^2 \tag{10.5}$$

where α is a parameter. There is no obvious choice for α. Say something about how α should be chosen.

There are two MCMC algorithms that can be easily implemented. We could use Metropolis-Hastings. The proposal distribution is to pick a circuit i at random (uniformly). Then we change x_i to $-x_i$. The other algorithm would be a Gibbs sampler. Note that the "dimension" is the number of circuits. Explain how each step of the MCMC only involves local computations.

10.2 Estimating derivative

We recall the network example from Kroese. The times T_i are independent and uniformly distributed but with different ranges: T_i uniform on $[0, \theta_i]$. Let U_1, U_2, U_3, U_4, U_5 be independent, uniform on $[0, 1]$. Then we can let $T_i = \theta_i U_i$. The random variable we want to compute the mean of is the minimum over all paths from A to B of the total time to traverse that path.

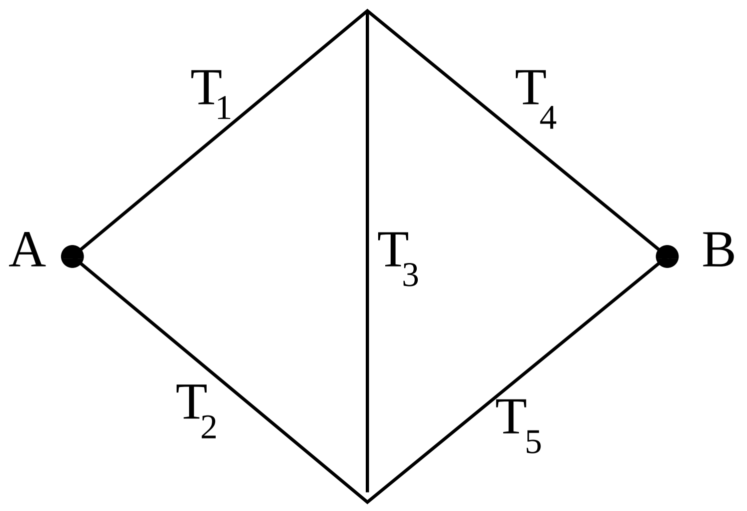

Figure 10.4: Network example. We seek the quickest path from A to B.

In the first figure we take
$\theta_2 = 2$,
$\theta_3 = 3$,
$\theta_4 = 1$,
$\theta_5 = 2$
and plot the mininum as a function of θ_1 using 10^6 samples. Two methods are used. For one we use different random numbers to generate the 10^6 samples for every choice of θ_1. For the other we use the same random numbers for different θ_1. The figure clearly shows that using "common" random numbers produces a much smoother curve. We should emphasize that the values of the min computed using the common random numbers are not any more accurate

than those computed using independent random numbers. It is just that the error go in the same direction.

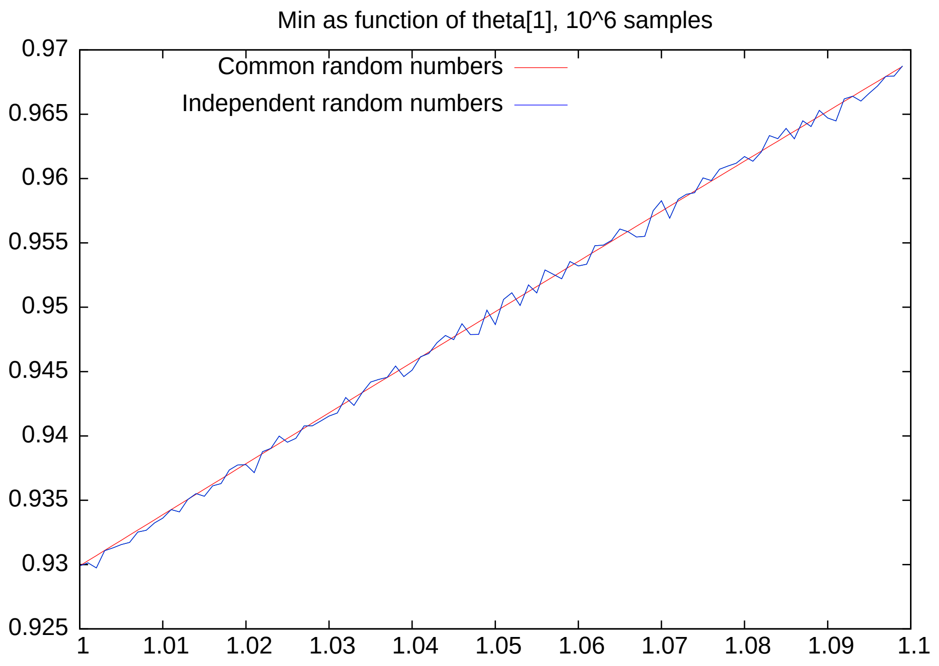

Figure 10.5: Network example. Plot of minimum as function of theta[1] using independent random numbers and common random numbers.

In figure (10.6) we compute a central difference approximation at $\theta_1 = 1$. We plot the approximation as a function of δ. The independent plots use independent samples of the random numbers. Two plots are shown, one using 10^6 samples, one with 10^7 samples. The other plot uses common random numbers. Clearly the common random number approach works much better. Starting around $\delta = 0.1$ the common random number curve starts to bend upwards, reflecting the error coming from the use of the central difference approximation. In these simulations we generate the random T_i by generating $U_1, \cdots, U_5$ uniformly in $[0, 1]$. Then we set $T_i = \theta_i U_i$.

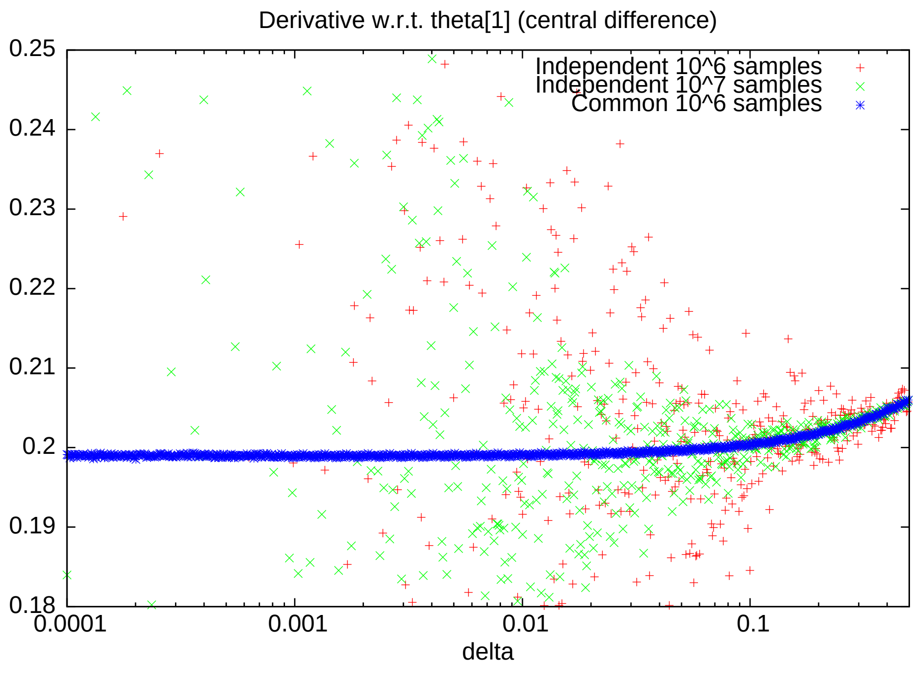

Figure 10.6: Network example. Plot of central difference approximation of derivative of minimum with respect to theta[1] using independent random numbers and common random numbers.

In figure (10.7) we continue to compute a central difference approximation to the derivative. The red and green plots are as in the previous figure. For the green plot we generate the T_i in a different way. We generate T_i using acceptance-rejection as follows. For each i we generate a random number in $[0, 5]$ and accept it when it is in $[0, \theta_i]$. We use common random numbers in the sense that we reset the random number generator to the original seed ...

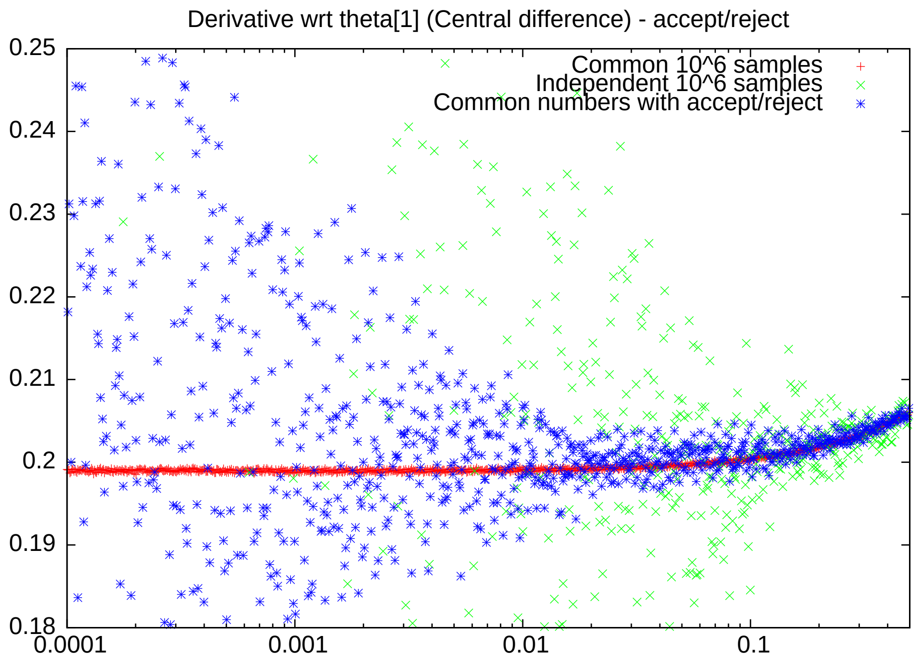

Figure 10.7: Network example. Plot of central difference approximation of derivative of minimum with respect to theta[1] using independent random numbers, good common random numbers and common numbers using accept/reject.

Finally in figure (10.8) we attempt to fix the accept-rejection approach. The idea is that common random numbers do not work well here since the number of attempts needed to accept can be different for $\theta_1 - \delta/2$ and $\theta_1 + \delta/2$. So we always generate 100 random numbers for each acceptance-rejection. Hopefully this keeps the common random numbers in sync. However, the result show in the figure is not really any better than the acceptance-rejection in figure (10.7).

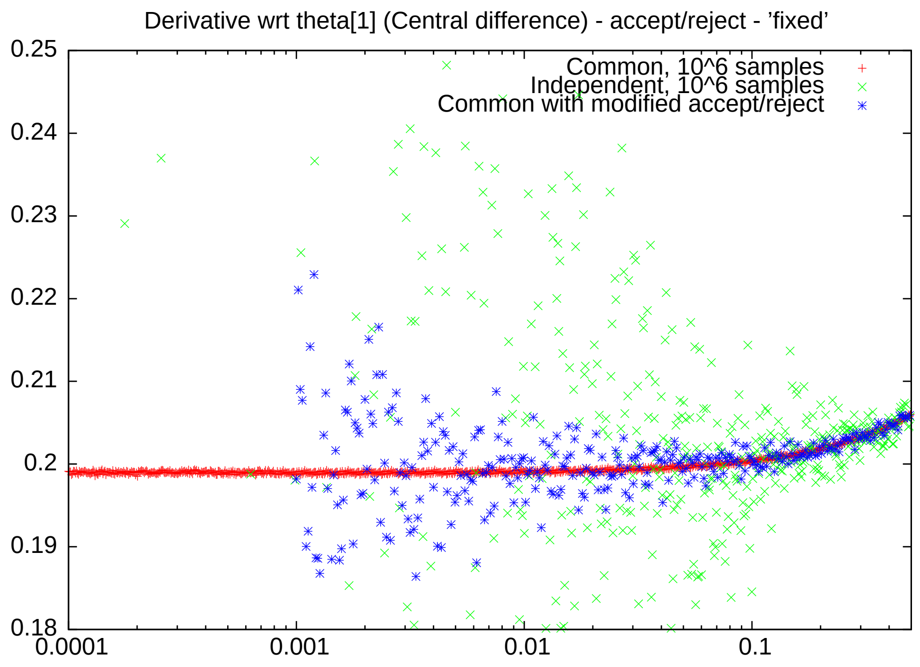

Figure 10.8: Network example. Plot of central difference approximation of derivative of minimum with respect to theta[1] using independent random numbers,good common random numbers and attemp to fix accept/reject.

10.3 Stochastic gradient descent