Your Homework Assigment
Monte Carlo Minimization and Counting:
One, Two, . . . , Too Many
Isabel Beichl, Dianne P. O'Leary, and Francis Sullivan

Problem 1. Consider the function `myf.m` (on the website), with domain $0 \le x \le 7$.
(a) Generate 500 uniformly distributed points on the interval $[0, 7]$ in Algorithm 1, using `fmincon` for the local minimizer. Make a graph illustrating the minimizer corresponding to each starting point.
(b) $L = 90.3$ is a Lipschitz constant for the function `myf.m`. Use Algorithm 2 on the interval $[0, 7]$. Compare the performance of the two methods.
(c) (Extra) Try Monte Carlo minimization on your favorite function of n variables for $n > 1$.

Answer: The programs `myfmin.m` and `myfminL.m` on the website solve this problem but do not make the graph. (If you find any bugs in `myfminL.m`, please let me know; it needs more testing.)

Problem 2. Use the simulated annealing algorithm to minimize `myf.m`. Experiment with various choices of T, α, and ϵ. Describe your experiment and the conclusions you can draw about how to choose parameters to make the method as economical and reliable as possible.

Partial Answer: The program `sim_anneal.m` on the website is one implementation of simulated annealing, and it can be run using `problem1_and_2.m`. To finish the problem, experiment with the program. Be sure to measure reliability as well as cost, and run multiple experiments to account for the fact that the method is randomized. Also comment on the number of runs that converge to $x = 1.7922$, which is a local minimizer with a function value not much worse than the global minimizer.

Problem 3.
(a) MATLAB provides a naïve Monte Carlo solution algorithm for the TSP. Given an ordering of the cities, it randomly generates a pair of cities and interchanges them if the interchange lowers the total distance. Experiment with the demonstration program `travel.m` and display the program using `type travel` to see how this works.
(b) Write a program to solve a TSP using simulated annealing, and compare your algorithm with that used in part (a).

Answer:
TBD

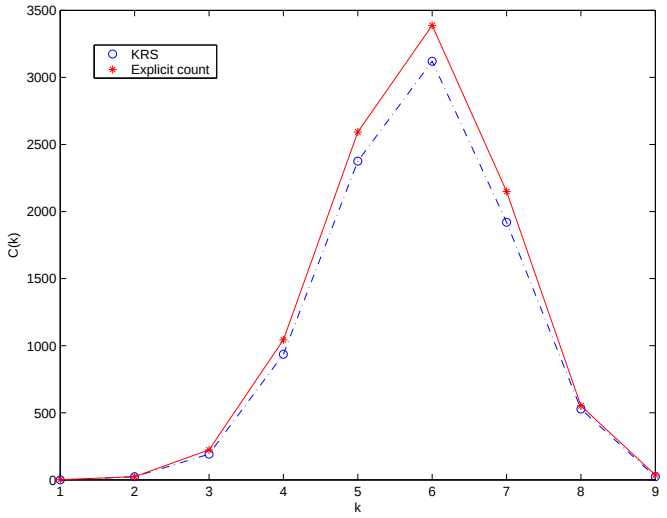

Figure 1: Counts obtained by the KRS algorithm and by explicit counting for a 4×4 lattice. For KRS we set the probabilities to 0.5, the number of steps between records to $\ell = 4$, and the total number of steps to 10^5. Because ℓ was so small, the samples were highly correlated, but the estimates are still quite good.

Problem 4.
(a) Compute the partition function coefficients for a 2×2 lattice by explicit counting. Repeat for a 3×2 lattice. Consider the 4×4 case and see why it is no longer reasonable to explicitly count the number of arrangements.
(b) Implement the KRS algorithm and use it to estimate the partition function for a 4×4 lattice. Try various choices of probabilities and updating intervals. Repeat for a lattice as large as possible (perhaps 10×10).

Partial Answer:

(a) Here are some explicit counts, some done by hand and some by `latticecount.m` by Thomas DuBois:

	$C(0)$	$C(1)$	$C(2)$	$C(3)$	$C(4)$	$C(5)$	$C(6)$	$C(7)$	$C(8)$
2×2	1	4	2						
2×3	1	7	11	3					
3×3	1	12	44	56	18				
4×4	1	24	224	1044	2593	3388	2150	552	36
6×6	1	60	1622	26172	281514	2135356	11785382	48145820	146702793

(b) One of the more interesting programming issues in this problem is the data structure.

- If we keep track of each edge of the lattice, then we need to enumerate rules for deciding whether two edges can be covered at the same time. For example, in our 2×2 lattice, we cannot simultaneously have a dimer on the top edge and one on the left edge.

- If we keep track of each node of the lattice, then we need to know whether it is occupied by a dimer, so our first idea might be to represent a monomer by a zero and a dimer by a 1. But we need more information – whether its dimer partner is above, below, left, or right. Without this additional information, the array

$$\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$$

  tells us that the 2×2 lattice has two dimers on it, but we can't tell whether they are horizontal or vertical.

- A third alternative is to keep track of both edges and nodes. Think of it as a matching problem: each node can be matched with any of its four neighbors in a dimer, or it can be a monomer. We maintain an array of nodes, where the jth value is 0 if the node is a monomer, and equal to k, if (k, j) is a dimer. We store the edges in an $n^2 \times 4$ array, where the row index indicates the node at the beginning of the edge, and the entry in the array records the node at the end. Thus, each physical edge has two entries in the array (in rows corresponding to its two nodes), and a few of the entries at the edges are 0, since some nodes have fewer than 4 edges. We can generate a KRS change by picking an edge from this array, and we update the node array after we decide whether an addition, deletion, or swap should be considered.

The program `KRS.m`, by Sungwoo Park, on the website, is an efficient implementation of the second alternative. Sample results are shown in Figure 1.

Please refer to the original paper [1] for information on how to set the parameters to KRS. Kenyon, Randall, and Sinclair showed that the algorithm samples well if both the number of steps and the interval between records are very large, but in practice the algorithm is considerably less sensitive than the analysis predicts.

[1] C. Kenyon, D. Randall, and A. Sinclair, Approximating the number of monomer-dimer coverings of a lattice J. Stat. Phys. 83, 637 (1996)