HA2 for Monte Carlo and Empirical Methods

Jingmo Bai Zuoyi Yu

February 21, 2023

1 Part1: Self-avoiding walks in $\mathbb{Z}^d$

1.1 Question 1

We want to prove $c_{n+m}(d) \leq c_n(d)c_m(d)$ and we are given

$$c_n(d) = |S_n(d)| \tag{1}$$

Assume S_n be all possible n steps self avoiding walks, S_m all possible m steps self avoiding walks, and S_{n+m} all possible n+m steps self avoiding walks. As we cannot go back to the step we have ever gone, the combination of n+m steps should be less than the sum of n and m steps separately. That is:

$$S_{n+m} \leq S_n \times S_m \tag{2}$$

so according to Equation 1, we can get what we want to prove:

$$c_{n+m}(d) = S_{n+m} \leq S_n \times S_m = c_n(d)c_m(d) \rightarrow c_{n+m}(d) \leq c_n(d)c_m(d) \tag{3}$$

for all $n \geq 1$ and $m \geq 1$.

1.2 Question 2

Firstly, take the logarithm of what we prove in the last part:

$$log(c_{n+m}(d)) \leq log(c_n(d)) + log(c_m(d)) \tag{4}$$

From the definition of subadditive, Equation 4 follows the rule:

$$a_{m+n} \leq a_m + a_n \tag{5}$$

so we can treat $log(c_n(d))$ as a subadditive. Therefore, the following limit and equation exits for $log(c_n(d))$:

$$\lim_{n\to+\infty} \frac{log(c_n(d))}{n} = inf_{n\geq 1} \frac{log(c_n(d))}{n} \tag{6}$$

Take the logarithm of what we want to prove:

$$log(\mu_d) = \lim_{n\to+\infty} \frac{1}{n} log(c_n(d)) \tag{7}$$

The right part is just what we get in Equation 6, so the limit $\mu_d = \lim_{n\to+\infty} c_n(d)^{\frac{1}{n}}$ exists.

1.3 Question 3

In this question, we are asked to estimate $c_n(2)'s$ with the sequential importance sampling (SIS) algorithm with instrumental distribution g_n. And g_n in this question is standard random walk $(X_k)_(^n k = 0)$ in $\mathbb{Z}^2$ which follows $X_0 = 0$ and each X_{k+1} is drawn uniformly among the four neighbours of X_k.

As it is a problem in $\mathbb{Z}^2$, for all d:

$$c_n(2) \sim A_2 \mu_2^n n^{\gamma_2 - 1} \tag{8}$$

And consider μ_d we get from the last part, Equation 8 can be also written as:

$$c_n(2) \sim \lim_{n \to +\infty} A_2 c_n(2) n^{\gamma_2 - 1} \sim \lim_{n \to +\infty} A_2 n^{\gamma_2 - 1} \tag{9}$$

Therefore, from the above equation, we can treat this problem as to simulate a large number N of random walks in $\mathbb{Z}^2$. We choose the large random walks number N = 10000 and 10 steps as example. As this is a problem in $\mathbb{Z}^2$, there are four possible moves from the current position: left, right, up and down. So in each loop, the primary start is at (0,0), we randomly choose a new direction for current position based on the former position and then store these new positions. Then check if this walk is self avoiding or not according to whether there are the same coordinates before, then count the number of self-avoiding walk to calculate the observed ration $\frac{N_{SA}}{N}$. As there are 4 possible directions for every n + 1 step, we can estimate $c_n(2)$ as:

$$c_n(2) = \frac{N_{SA}}{N} * 4^n \tag{10}$$

The estimated c_n and the true value of c_n is shown in Table 1. Compared with the true value found in Self-Avoiding Walks[1], this estimator is precise when step is small, about when *steps* $\leq$ 6, and the difference between estimated values and true values becomes larger when steps are large. So we can conclude that it is hard to find a self-avoiding random walk with large steps.

Table 1: The values of our estimated c_n and true c_n with sequential importance sampling algorithm

Steps	Estimated c_n	True c_n
1	4	4
2	12	12
3	36	36
4	100	100
5	285	284
6	782	780
7	2191	2172
8	6311	5916
9	17066	16268
10	45508	44100

1.4 Question 4

In this question, we will improve the results with replacing g_n with self-avoiding walks in $\mathbb{Z}^2$ starting in the origin. Therefore, instead of choosing random directions like Q3, we add weights

and a check code of visited positions for self-avoiding walks method. And estimate c_n using the following equation from the slides for Lecture 7[2]:

$$c_n \approx \frac{1}{N} \sum_{i=1}^{N} \omega_N^i$$

To be specific, we create a grid for different steps and assume the origin at the center of this grid, with visited coordinates as 1 and possible moves coordinates as 0. We set the initial weights as 1 and create a flags array to store the weights of all iterations for different steps. We also create an XY array to store all the movements for different steps. In each loop, the movements start from the origin. Then, to achieve self-avoiding walks idea, we find the neighbours with a number 0 on the grid, since 1 means this position has been visited before so we cannot go back there again, then choose a direction randomly. Set the new weights as the product of current weights and the sum of neighbours. Repeat these steps until reach the objective steps. During this loop, if the neighbours of the current position are all 1, which means there are no possible moves in this position, store the weights and break this loop. After all steps are made, we check if this is a self-avoiding walks according to check the last row of XY array: if it is (0,0), it means the loop breaks ealier due to no possible moves, so this is not a self-avoiding walk. Store all the weights of self-avoiding walks in each iteration for different steps. The final estimated steps is the mean of these weights.
We choose steps = 15 and N = 10000 in this problem, the results we get are shown in Table 2.

Table 2: The values of our estimated c_n and true c_n using self-avoiding walks method

Steps	Estimated c_n using SAW	True c_n
1	4	4
2	12	12
3	36	36
4	100	100
5	284	284
6	781	780
7	2172	2172
8	5890	5916
9	16269	16268
10	43978	44100
11	121361	120292
12	327365	324932
13	890252	881500
14	2386469	2374444
15	6554686	6416596

From Table 2, we can see a great improvement of accuracy with SAW methods compared with standard random walks. Our estimated values are quite the same as the true values when steps ≤ 10, and the differences for large steps are not quite big as well.

1.5 Question 5

We are required to use sequential importance sampling with resampling (SISR) in this task. The main idea of SISR is to replace the small weights with the large weights. A common method is to

replace new particles $\tilde{X_1^{0:n}}, ..., \tilde{X_1^{0:n}}$ with $X_1^{0:n}, ..., X_N^{0:n}$ with probabilities given by the normalized importance weights[2].This can be done with using the MATLAB function *randsample*.

The general idea of this problem is very similar to Question 5. We start from the center of the grid and set the initial weight as 1. Then we check all the possible moves from the current position and make a random movements. The new weight is the sum of the possible moves among neighbours. We also check if it is a self-avoiding walk according to whether there is any 0 in the neighbours of the current positions. Besides these, we also add some other details for SISR methods. Instead of generating new grid and move recording for each iteration, we create a cell to store all the grids, weights and movements for each iteration of each step. As the estimation of c_n using SISR is[2]

$$c_{N,n} = \prod_{k=0}^{n} (\frac{1}{N} \sum_{i=1}^{N} \omega_k^i) \tag{11}$$

we record weights in an array instead of just recording the total number, then use Equation 11 to calculate c_n. Then, we use the function *randsample* to resample and update grids and movements of walk based on weights.

We still choose steps = 15 and N = 10000. Compared the estimated c_n in Table 3 with values in Table 2, we can find both these two methods have good estimation for small steps, but SISR works worse than SIS for large steps. So we can say SISR does not fit this task very well.

Table 3: The values of our estimated c_n and true c_n using SISR method

Steps	Estimated c_n using SISR	True c_n
1	4	4
2	12	12
3	36	36
4	100	100
5	284	284
6	778	780
7	2179	2172
8	5853	5916
9	16306	16268
10	44190	44100
11	120240	120292
12	328015	324932
13	891874	881500
14	2403599	2374444
15	6484911	6416596

1.6 Question 6

To estimate the parameters in Equation 8, we can logarithm of it first:

$$\begin{aligned} log(c_n(2)) &= log(A_2 \mu_2^n n^{\gamma_2 - 1}) \\ &= log(A_2) + nlog(\mu_2^n) + (\gamma_2 - 1)log(n) \end{aligned} \tag{12}$$

We can treat Equation 13 as a linear regression equation with the form

$$z = ax + by + c \tag{13}$$

where

$$\begin{aligned} x = n, y = log(n) \\ a = log(\mu_2), b = \gamma_2 - 1, c = log(A_2) \end{aligned} \tag{14}$$

This linear regression problem can be easily solved by the built-in method in MATLAB using '\'. Then we get the following estimation:

$$a = 0.9888, b = 0.1838, c = 0.3944 \tag{15}$$

Then plug these coefficients into Equation 14, the final values of estimated parameters we want are shown in Table 4. We redo the estimation several times.

Table 4: The estimated values of parameters A_2, μ_2, γ_2

Times	A_2	μ_2	γ_2
1	1.4834	2.6880	1.1838
2	1.4977	2.6608	1.2072
3	1.4907	2.6603	1.2153
4	1.4804	2.6636	1.2228
5	1.4919	2.6657	1.2063

Then analyse this problem in theory. We know the true value of γ_2 is $\frac{43}{32}$, so $n^{\gamma_d - 1} < n$. Consider Equation 8 again:

$$c_n(2) \sim A_2 \mu_2^n n^{\gamma_2 - 1}$$

A_d is a constant, μ_d^n is a exponential function and $n^{\gamma_d - 1} < n$. So μ_d^n has the biggest impact on the final result while changing variable n, which means μ_d^n should be the most easy parameter to estimate.

Then see Table 4, as we know the true value of $\gamma_2 = 1.344$, while our estimation value is about 1.210, so our estimation is not that accuracy. What's more, among these experiments, we can find mu_2 has the most steady estimation while A_2 and γ_2 vary a lot, which means our assumption: mu_2 is the easiest parameter to estimate, is true.

1.7 Question 7

We know that

$$\mu_d = \lim_{n \to +\infty} c_n(d)^{\frac{1}{n}} \tag{16}$$

The number μ_d could be interpreted as the geometric mean of the number of un-visited neighbours (sequentially looking one step ahead) along a self-avoiding path.
First, we assume that we only take a walk heading towards the positive directions along all the coordinates in any dimension d. Then for each step, we have d choices. Thus, for a walk length of n, the amount of possible walks is d^n. This conditional assumption will clearly satisfy that all these d^n walks are self-avoiding walks. So the number of self-avoiding walks $C_n(d) \geq d^n$. Second, we assume that we can take a walk along both the positive and negative directions of all the coordinates in any dimension d, but not go back to the previous step. Thus we have $2d$ choices for the first step and $2d - 1$ choices for all the rest steps. The total amount of possible walks is $2d(2d-1)^{n-1}$. It's also clear that the number of self-avoiding walks $C_n(d) \leq 2d(2d-1)^{n-1}$. So, we have obtained that

$$d^n \leq C_n(d) \leq 2d(2d-1)^{n-1}$$

$$d \leq C_n(d)^{\frac{1}{n}} \leq (2d(2d-1)^{n-1})^{\frac{1}{n}} \tag{17}$$

When $n \to +\infty$,

$$\lim_{n\to+\infty} c_n(d)^{\frac{1}{n}} = \mu_d$$

$$\lim_{n\to+\infty} (2d(2d-1)^{n-1})^{\frac{1}{n}} = \lim_{n\to+\infty} (\frac{2d}{2d-1})^{\frac{1}{n}}(2d-1) = 2d-1$$

Thus, the following general bound should hold.

$$d \leq \mu_d \leq 2d-1 \tag{18}$$

1.8 Question 8

We know that for $d \geq 5$

$$c_n(d) \sim A_d \mu_d^n n^{\gamma_d - 1} \tag{19}$$

And it is also known that $\gamma_d = 1$ for $d \geq 5$.
Inserting (14) into (3) yields

$$A_d \mu_d^{n+m}(n+m)^{\gamma_d-1} \leq A_d \mu_d^n (n)^{\gamma_d-1} A_d \mu_d^m (m)^{\gamma_d-1}$$

$\gamma_d = 1$

$$A_d \mu_d^{n+m} \leq A_d^2 \mu_d^{n+m}$$

$$A_d^2 \geq A_d$$

$$|A_d| \geq 1$$

We know that c_n is the number of possible walks so it's a non-negative number, and the connective constant μ_d is also positive, and the walk of length n is also positive. So we can conclude that A_d is a non-negative number. Thus, the following general bound should hold, for $d \geq 5$

$$A_d \geq 1$$

1.9 Question 9

In this part, we need estimate A_d, μ_d and γ_d for $d \geq 3$. The general idea for larger d is quite the same as what we do in Question 6, and we just modify the dimension of grids and small details in the code.

To compare the bounds in the last two questions, we choose d = 5 and N = 10000, the corresponding parameters A_5, μ_5 and γ_5 is

$$A_5 = 1.1310, \mu_5 = 8.8321, \gamma_5 = 1.0334 \tag{20}$$

Then we verify the bounds from Question 7 first:

$$d = 5 \leq \mu_5 = 8.7803 \leq 2 * 5 - 1 = 9 \tag{21}$$

Therefore, we can find our estimation μ_5 is inside the bound in Question 7 when d = 5. Then, from problem 8, we know that when $d \geq 5$, our $A_d = 1.1473 \geq 1$, So our estimation μ_5 and A_5 are both follow the boundary rules above.

Then, compare our μ_d for large d with asymptotic bound found in Graham[3]. The formula of asymptotic bound on μ_d is

$$\mu_d \sim 2d - 1 - \frac{1}{2d} - \frac{3}{(2d)^2} - \frac{16}{(2d)^3} + O(\frac{1}{d^4}) \tag{22}$$

So this can be calculated easily with MATLAB. We choose d from 6 to 10 and N = 10000. The results are shown in Table 5.

Table 5: The estimated and asymptotic bound values of parameter μ_d

steps	estimated μ_d	asymptotic bound μ_d
6	10.8617	10.8866
7	12.8835	12.9074
8	14.9000	14.9219
9	16.9114	16.9324
10	18.9215	18.9405

From the above table, we can see our estimated values are quite close to the asymptotic bounds. So we can say SISR is a good method to estimate parameters.

2 Part2: Filter estimation of noisy population measurements

In this part, our task is to estimate the relative population size in generation k of some organism. We model it as a hidden Markov model where the relative population size X_k is the hidden Markov chain. In a simplified model we have the following dynamics as transition density.

$$X_{k+1} = B_{k+1}X_k(1 - X_k), B_{k+1} \in U(0.9, 3.9), k = 0, 1, 2, ..$$

We assume that the initial distribution

$$X_0 \in U(0.6, 0.99)$$

Due to problems of measuring the exact population, we get a measurement with the following observation density.

$$Y_k|X_k = x \in U(0.7x, 1.2x)$$

To implement the SISR algorithm, we need to modify the code on slide 24 of Lecture 7. To be specific, the observation density, the mutation, and the initialization part.
We first define the observation density, then initialize X_0 and the first estimation and weight, and then we select random samples. Afterwards, in 50-times iterations, the mutation or transition density is defined, the weight is updated, the filter expectation tau is estimated, and then the selection is done by resampling with replacement.
Figure 1 shows the estimation of the filter expectation τ_k and the true X value for 50 generations. In general, the estimation follows the true value, even though there are some missed peaks.

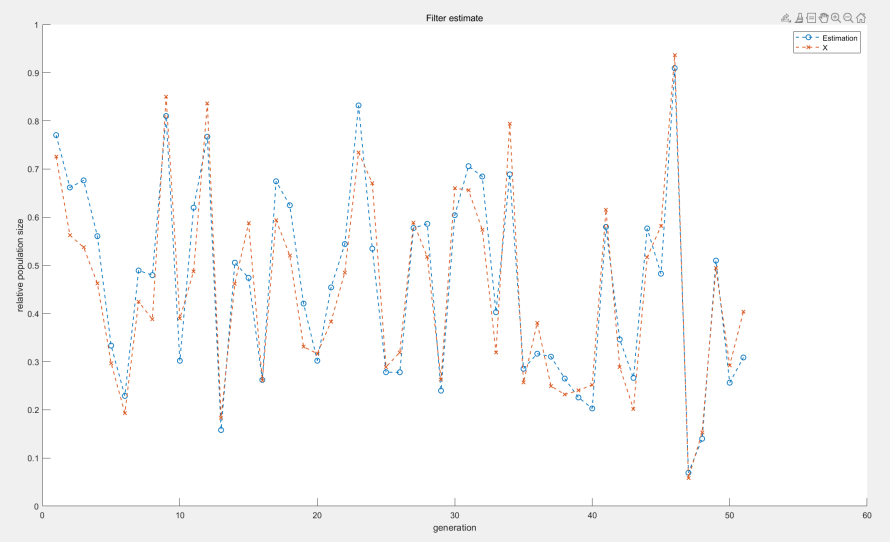

Figure 1: The estimation of the filter expectation and the true X

We also want to compute a 95% point wise confidence interval for the hidden state at time k using the weighted sample. We do it according to the code on slide 25 of Lecture 7. Figure 2 shows the 95% confidence interval for X_k and the true X value for 50 generations. In most cases, the true value is inside the confidence interval. However, for k = 9, 39, 49, 50, the true value falls outside the lower and upper limits.

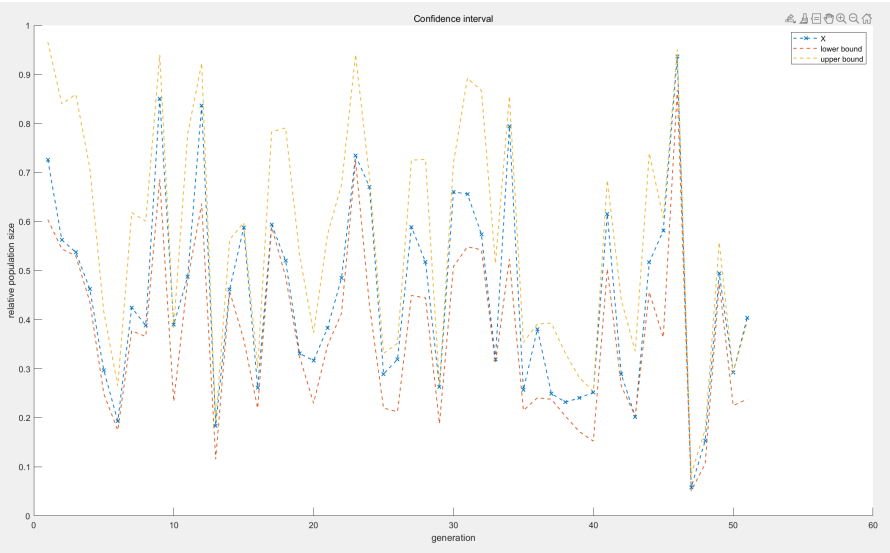

Figure 2: The confidence interval for X_k and the true X

At last, we can conclude that by adapting the SISR algorithm we can get quite good estimations of the relative population size in generation k in the question.

3 Reference

1. Gordon Slade. (1994)Self-Avoiding Walks. Available at: https://personal.math.ubc.ca/ slade/intelligencer.pdf
2. Magnus Wiktorsson, Lecture slides, FMSN50, Monte Carlo and Empirical Methods for Stochastic Inference, Lund University
3. Graham, BT. (2018) Borel-type bounds for the self-avoiding walk connective constant, Available at:http://arxiv.org/pdf/0911.5163.pdf

A Value in question 10

X	Lower bound	Upper bound
0.5625	0.5427	0.8173
0.5373	0.5301	0.8376
0.4627	0.4363	0.7134
0.2969	0.2522	0.4059
0.1933	0.1779	0.2619
0.4245	0.3755	0.6091
0.3880	0.3630	0.5946
0.8501	0.6825	0.9328
0.3896	0.2353	0.3735
0.4878	0.4780	0.7848
0.8365	0.6382	0.9340
0.1837	0.1153	0.1874
0.4620	0.4570	0.5652
0.5874	0.3628	0.5872
0.2615	0.2195	0.3002
0.5941	0.5877	0.7636
0.5201	0.4910	0.7976
0.3310	0.3252	0.5332
0.3167	0.2309	0.3735
0.3832	0.3486	0.5738
0.4851	0.4087	0.6754
0.7347	0.7278	0.9359
0.6705	0.4273	0.6701
0.2890	0.2063	0.3357
0.3192	0.2109	0.3469
0.5888	0.4526	0.7229
0.5176	0.4420	0.7248
0.2630	0.1928	0.2668
0.6603	0.5085	0.7048
0.6561	0.5495	0.8671
0.5738	0.5414	0.8841
0.3192	0.3106	0.5082
0.7944	0.5249	0.8576
0.2566	0.2168	0.3541
0.3807	0.2417	0.3918
0.2494	0.2382	0.3923
0.2319	0.2006	0.3299
0.2401	0.1694	0.2755
0.2520	0.1532	0.2526
0.6155	0.5019	0.6690
0.2887	0.2642	0.4311
0.2017	0.2010	0.3344
0.5170	0.4532	0.7296
0.5813	0.3663	0.5973
0.9365	0.8705	0.9556
0.0583	0.0539	0.0847
0.1536	0.1062	0.1761
0.4942	0.4764	0.5568
0.2923	0.2250	0.2892
0.4039	0.2371	0.3938