Computational Physics
Problem Set 5

Saleh Shamloo Ahmadi
Student Number: 98100872

October 25, 2021

1 2D Random Walk

As described in the lecture notes, we expect

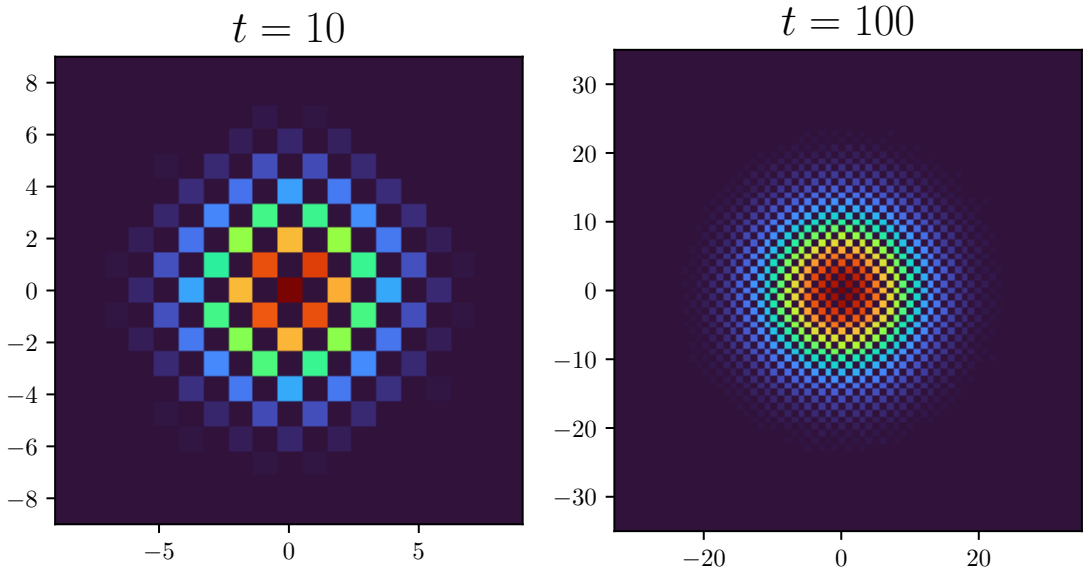
$$\langle r^2 \rangle = 2dDt \tag{1}$$

where d is the number of dimensions (degrees of freedom), t is the time the walker has spent moving and

$$D = \frac{l}{2d\tau} \tag{2}$$

where l and τ are the space and time steps respectively. In our simulations, units are normalized such that both l and τ are equal to 1. Hence, we expect

$$\langle r^2 \rangle = t, \tag{3}$$



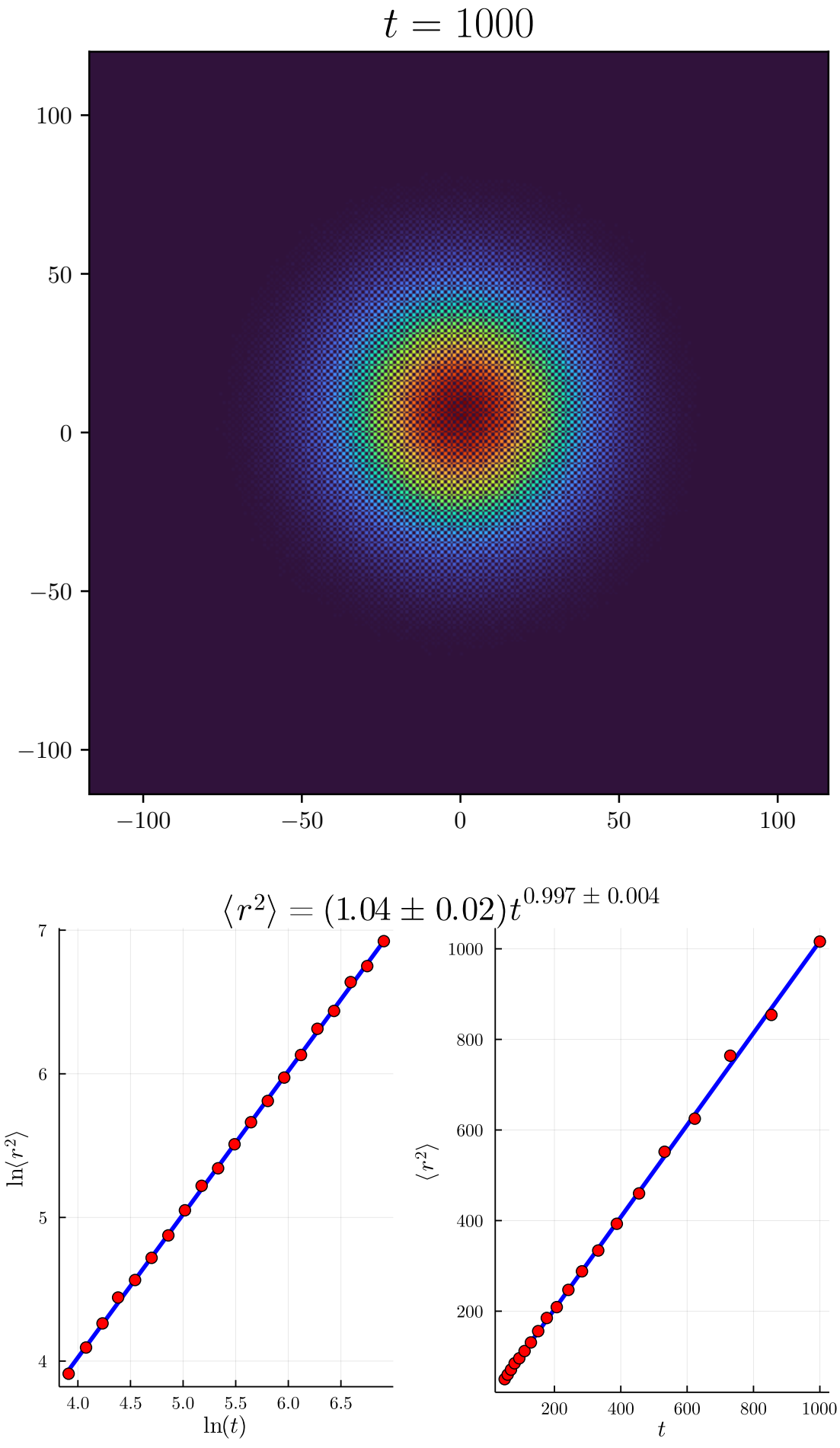


Figure 1: The results of the simulations are consistent the theoretical analysis

2 Diffusion-Limited Aggregation (DLA)

In this process, particles undergoing a random walk stick together to form clusters. The generated shapes appear in many systems in nature. DLA can describe aggregation in any system where diffusion is the primary means of transport in the system. You can read more about DLA on Wikipedia and Paul Bourke's blog.

The particles start moving from outside the boundaries of the shape and get stuck as soon as they come in contact with the cluster. The system starts growing from a "seed", which consists of the starting particles.

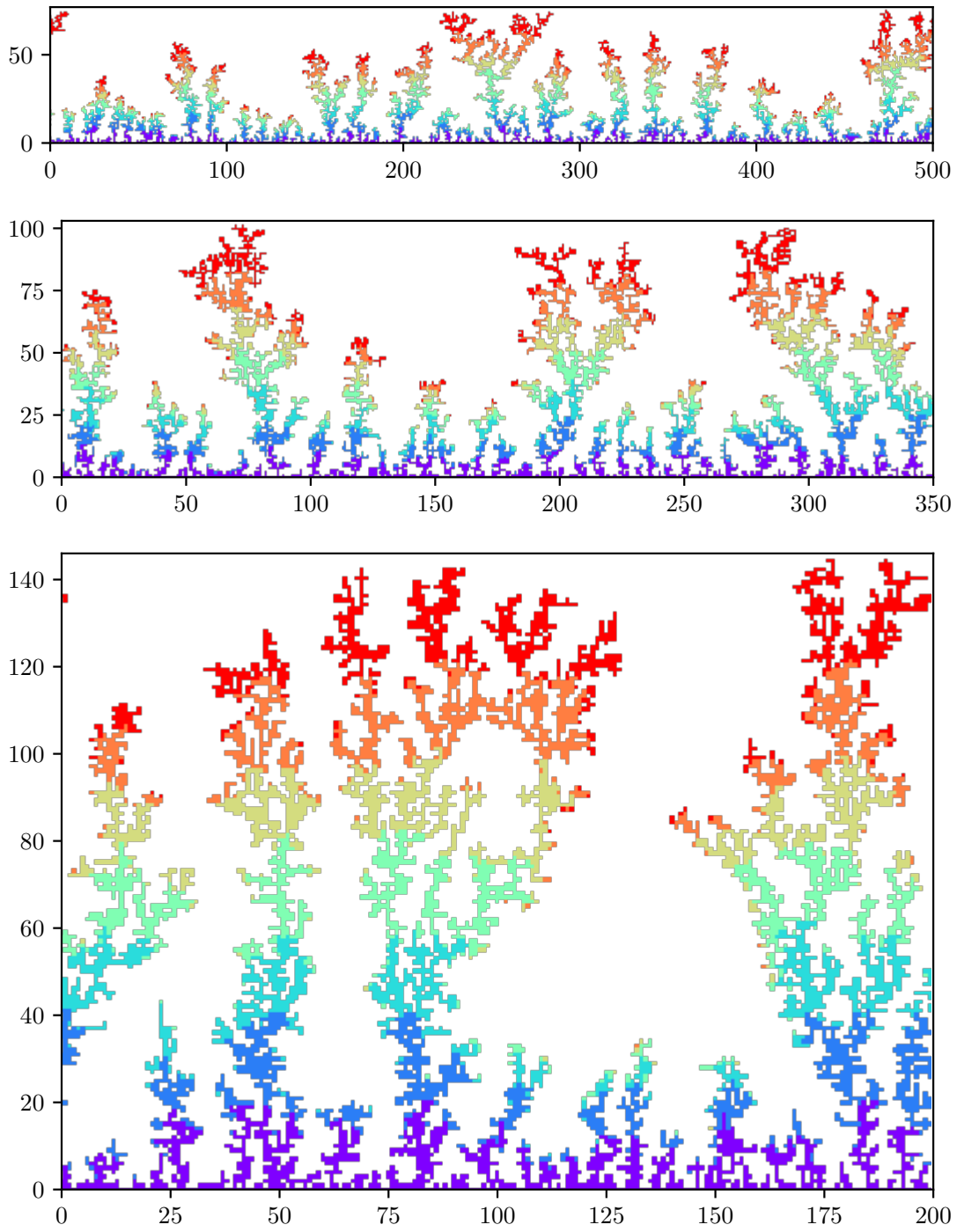


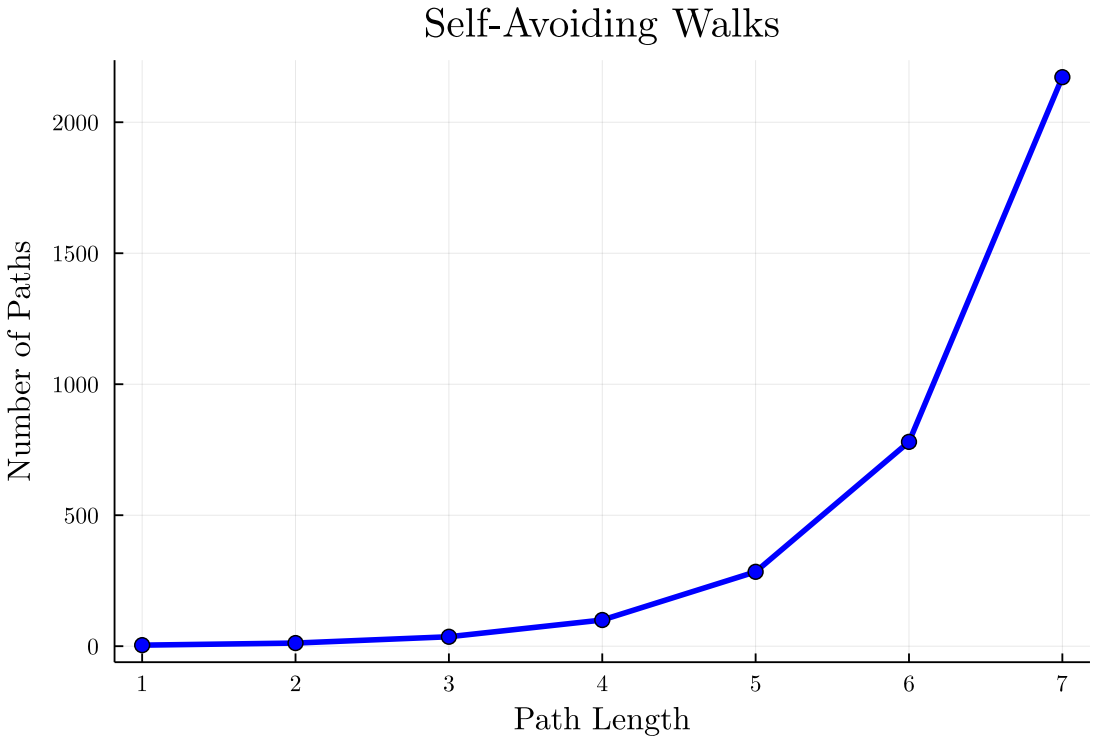
Figure 2: DLA with periodic boundary conditions and a line seed. Particles start moving form the top of the cluster.

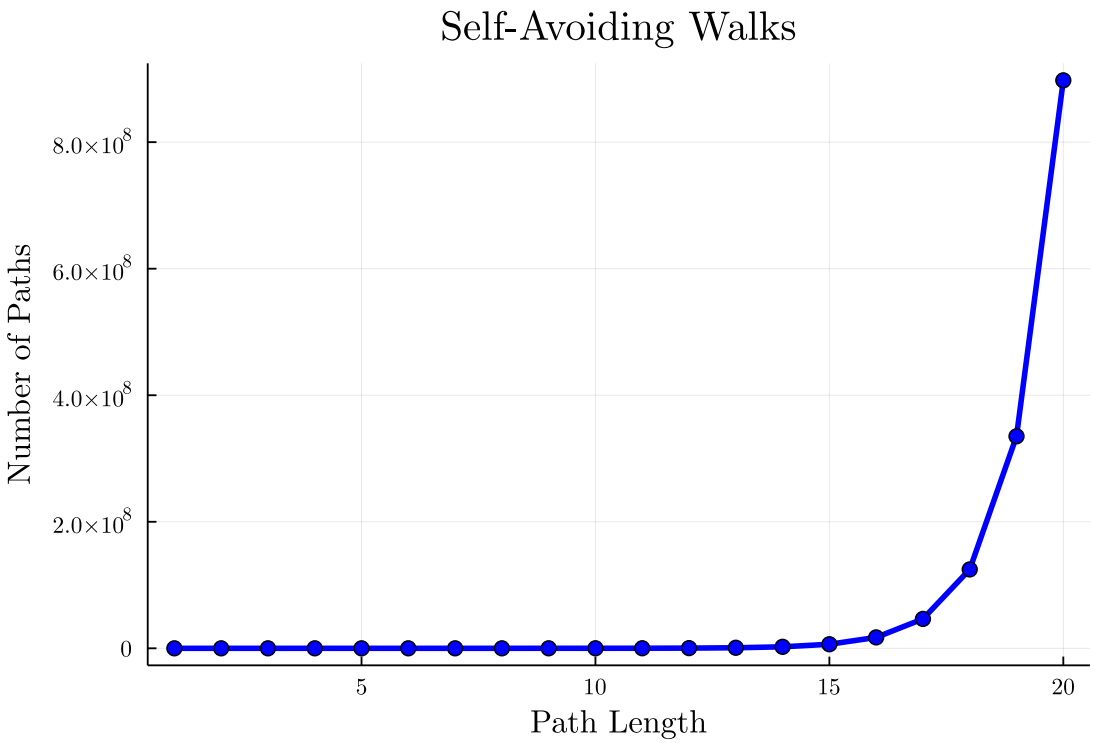
3 Self-Avoiding Walk (SAW)

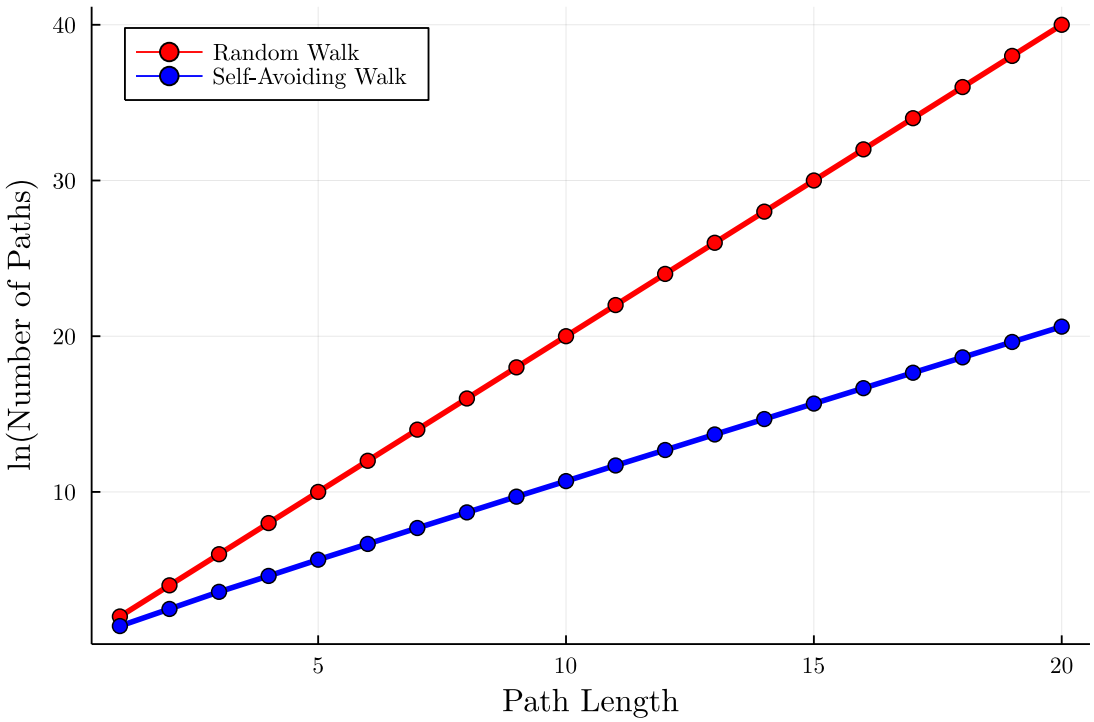
In this type of walk, the path of the walker cannot cross itself. The problem of counting the number of possible paths of a given length is NP-hard, but it can be approximated with good accuracy.

Algorithm 1 Recursively Counting Every Possible Self-Avoiding Walk of Length N in a Graph G

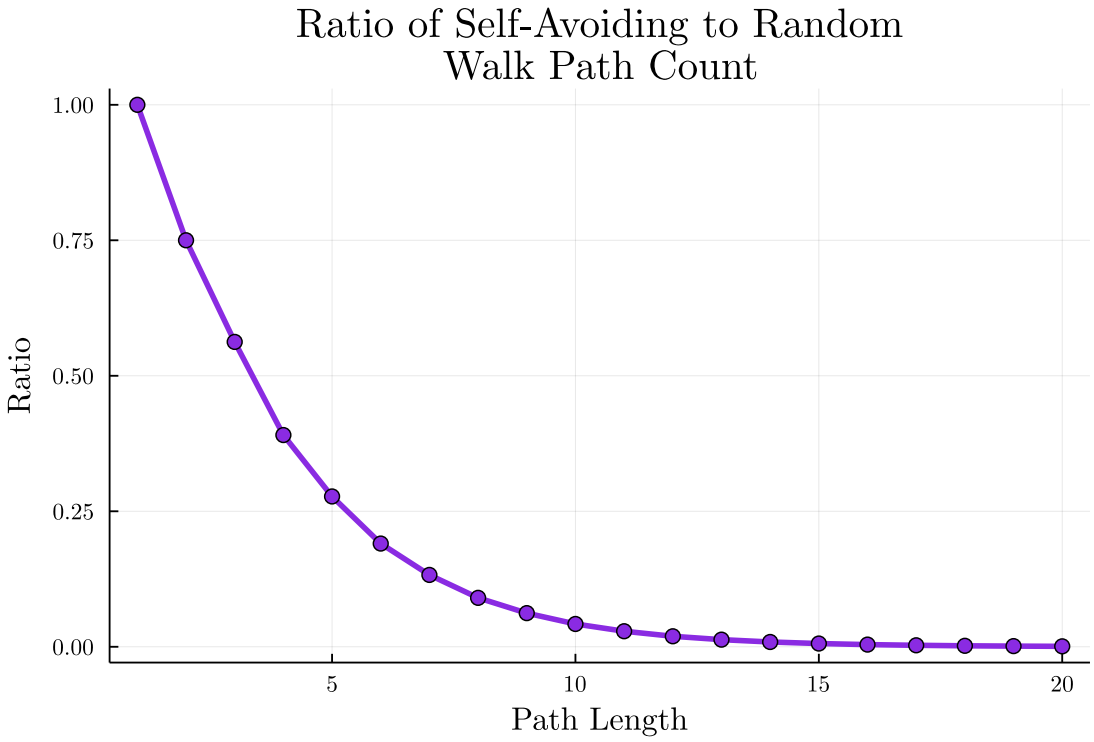
```
1: function CountSAW(N, G, p)                    ▷ p is the starting position
2:     if N = 0 then return 1
3:     else
4:         mark p as passed
5:         count ← 0
6:         for all q ∈ G.neighbors(p) do
7:             if q is not passed then
8:                 count ← count + CountSAW(N − 1, G, q)
9:             end if
10:        end for
11:        mark p as unpassed                    ▷ This step is called "backtracking"
12:        return count
13:    end if
14: end function
```







As you can see, the growth is consistently exponential. This is explored in more detail in the lecture notes.


Ratio of Self-Avoiding to Random Walk Path Count
1.00
0.75
0.50
0.25
0.00
Ratio
5
10
15
20
Path Length


Ratio of Self-Avoiding to Random Walk Path Count
0
−2
−4
−6
ln(Ratio)
5
10
15
20
Path Length

4 Random Number Generator (RNG)

I use the Julia programming language for solving these problems. The default random engine on Julia is a Mersenne Twister, so that is the engine that I analize here.

4.1 Distribution

The bare minimum requirement for an RNG is a uniform output distribution.

4.2 Coefficient of Variation

For random distributions, the coefficient of variation must be proportional to the inverse square root of the number of samples.

$$CV = \frac{\sigma}{N} \sim \frac{1}{\sqrt{N}} \tag{4}$$

Note that this is exactly the same as random ballistic distribution, which we explored before.

4.3 Numbers Coming Before 4

We can also analize the behavior of an RNG in other ways; If we take every number coming before a 4 in a random series generated by the RNG, the results must stay the same.

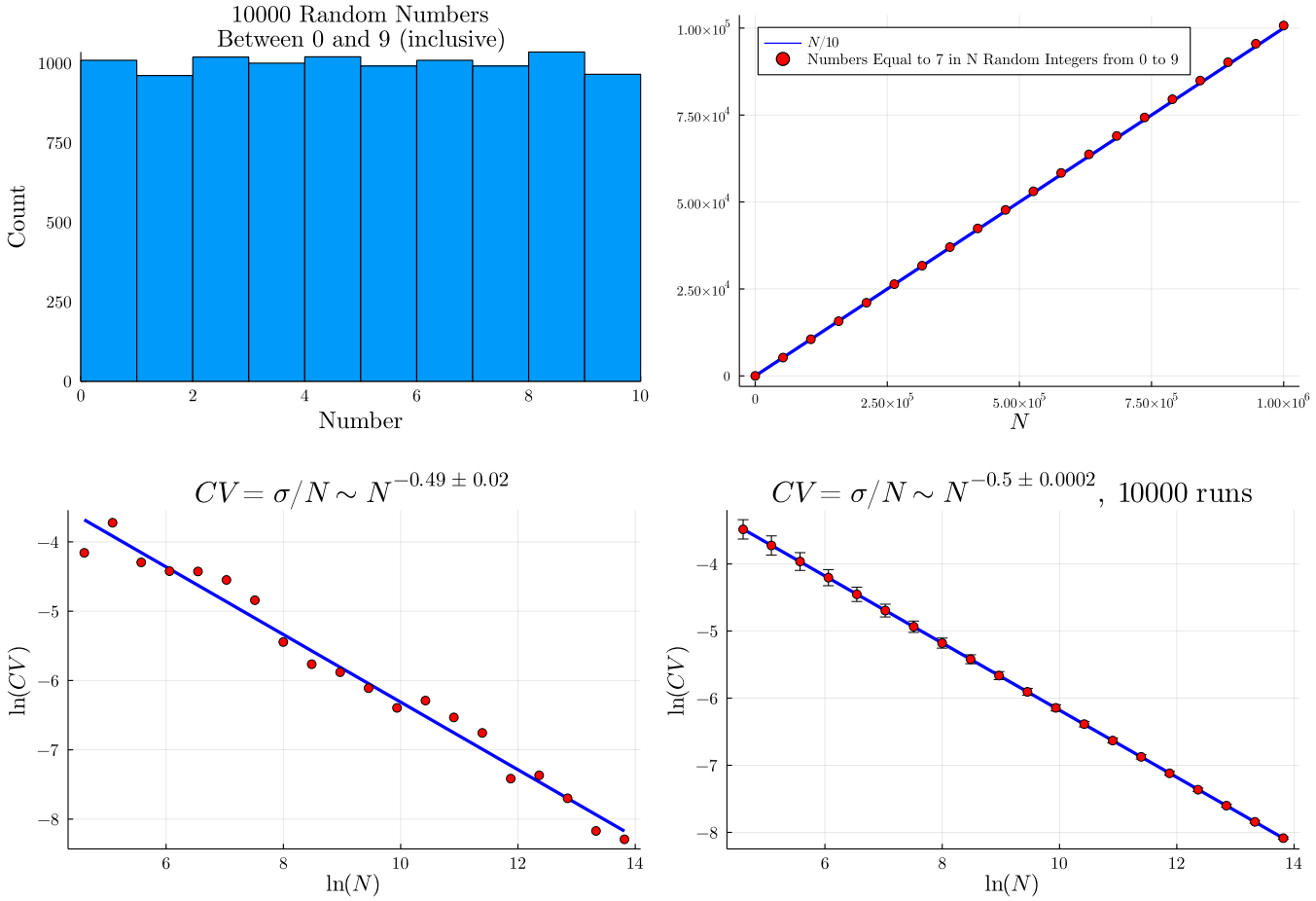


Figure 3: The distribution of numbers is fairly uniform, as expected.
The numbers are more evenly distributed as the number of random samples increases.

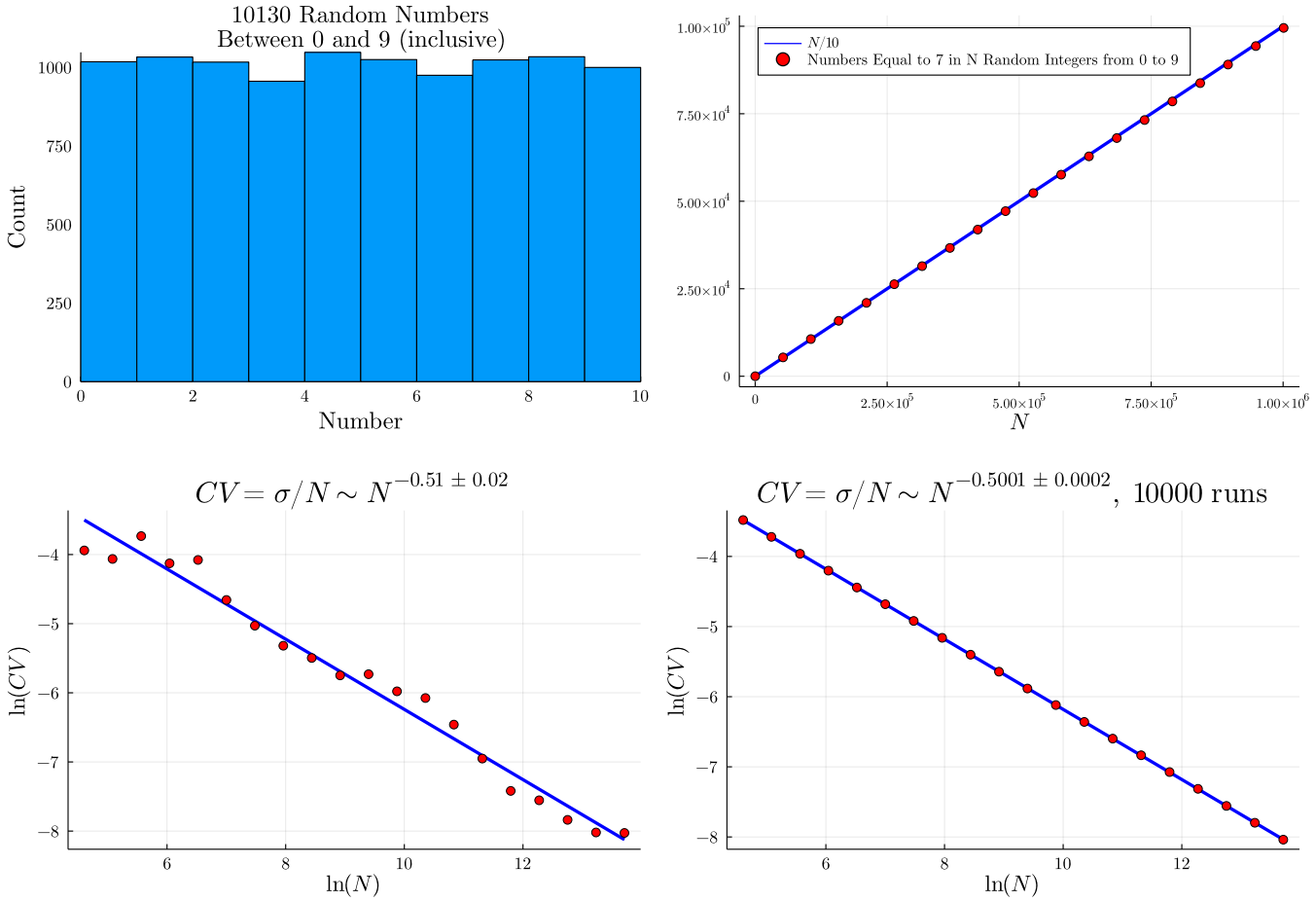


Figure 4: Taking numbers before 4 in a series gives the same results, as expected