Random walks and its applications

Mauricio Böckin, mbockin@kth.se

October 22, 2021

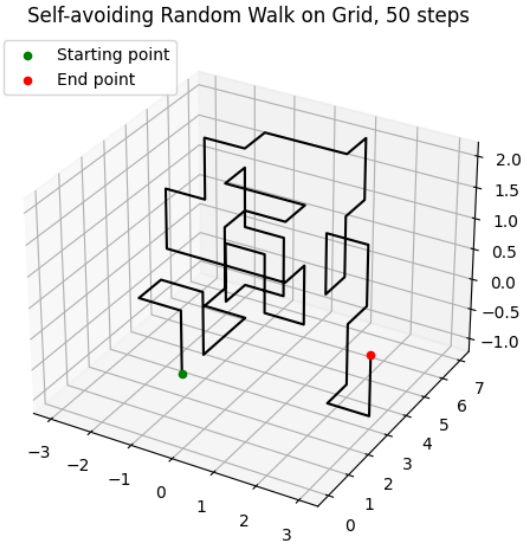
Contents

1 Introduction

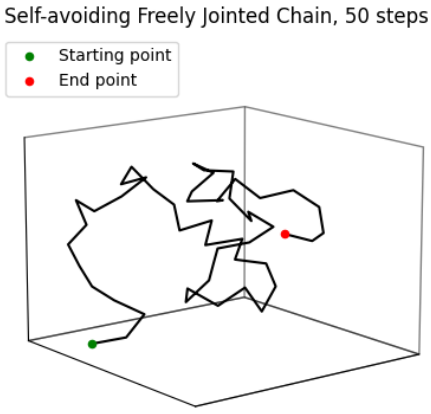
1.1 Project formulation

A random walk is a stochastic process that describes a path that consists of a succession of random steps on some mathematical space[1]. In this project different types of random walks in $\mathbb{R}^3$ are studied.

3D-random-walks on a grid and 3D-random-walks on a chain with fixed lengths links with random orientations, also known as freely jointed chain walks, are simulated. Both non-self-avoidng walks and self-avoiding walks are considered.



(a)



(b)

Figure 1: (a): Self-avoiding walk on a grid (b): Self-avoiding freely jointed chain walk

The end-to-end vector $\mathbf{R}$ is defined as $\mathbf{R}=\sum_{i=1}^{n} \mathbf{b}_i$ where $\mathbf{b}_i$ is the translation vector. The norm of $\mathbf{R}$, the end-to-end distance R, is simply the distance between the starting point and the end point of a walk and is the quantity of interest in this project, i.e. the analyzed quantity.

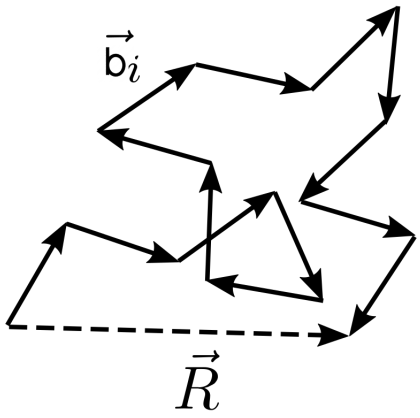


Figure 2: Source: [2]

1.2 Application: Polymers

Polymers are long molecules composed of repeating units known as monomers. Polymers in which there are no interactions (no potential differences) between monomers are called ideal polymers. Ideal polymers can be modelled by random walks since the orientation of a monomer, which in a random walk is analogous to a step, is independent of the other monomers in a polymer.

When modelling polymers the usual quantity of interest is the root mean square end-to-end distance of R, $\sqrt{< R^2 >}$, since it can be related to the radius of gyration (the average squared distance of any point from it's center of mass) of a polymer[2].

2 Discretisation and simulation

2.1 Non-self-avoiding 3D random walk on grid

This random walk was implemented by using the following algorithm:

- Initiate a walk by creating a positional vector $\mathbf{r} \in \mathbb{R}^3$ in the origin.
- Generate a random integer in range [0,5] from a uniform probability distribution.
- Construct a vector $\mathbf{b}$ of length b in the $\pm x$, $\pm y$ or $\pm z$ direction depending on the number generated.
- Take a step by setting $\mathbf{r} = \mathbf{r} + \mathbf{b}$

2.2 Self-avoiding 3D random walk on grid

This random walk was implemented by using the following algorithm:

- Initiate a walk by creating a positional vector $\mathbf{r} \in \mathbb{R}^3$ in the origin.
- Generate a random integer in range [0,5] from a uniform probability distribution.
- Construct a vector $\mathbf{b}$ of length b in the $\pm x$, $\pm y$ or $\pm z$ direction depending on the number generated.
- Construct a vector $\mathbf{r}_{temp} = \mathbf{r} + \mathbf{b}$
- Check if $\mathbf{r}_{temp}$ has not been visited previously in the walk. If this is the case continue to the next item in this algorithm, otherwise go back to item 2.
- Take a step by setting $\mathbf{r} = \mathbf{r} + \mathbf{b}$

2.3 Non-self-avoiding freely jointed chain walk

This random walk was implemented by using the following algorithm:

- Initiate a walk by creating a positional vector $\mathbf{r} \in \mathbb{R}^3$ in the origin.
- Generate a vector $\mathbf{b} \in \mathbb{R}^3$ of length b with a random orientation using an appropriate probability distribution.
- Take a step by setting $\mathbf{r} = \mathbf{r} + \mathbf{b}$

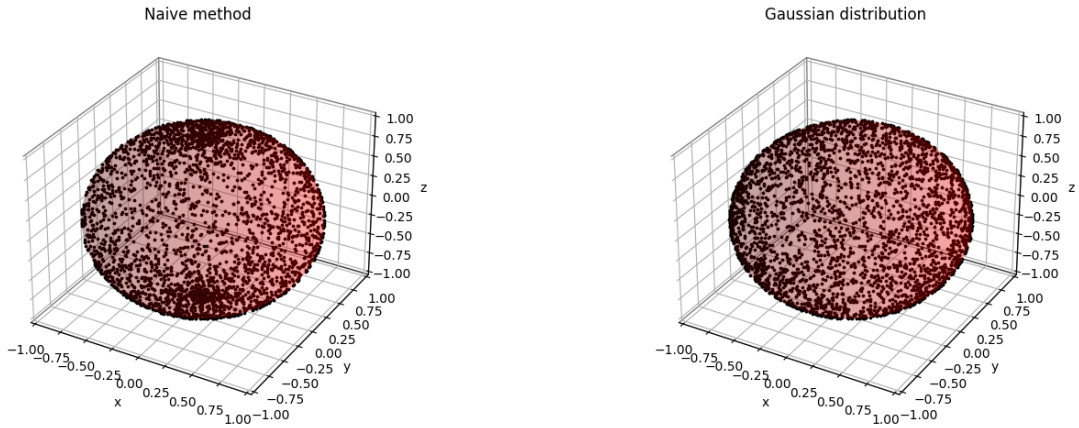
It may be worth taking a more detailed look at item two. In this part of the algorithm one has to generate a vector of length b with a random orientation, i.e. one has to be able to generate uniformly distributed points on a sphere of radius b.

It may be tempting to simply use spherical coordinates

$$x = rsin(\theta)cos(\phi)$$
$$y = rsin(\theta)sin(\phi)$$
$$z = rcos(\theta)$$
$$r \in [0, \infty), \theta \in [0, \pi], \phi \in [0, 2\pi]$$

and generate a uniform distribution of θ and ϕ whilst setting $r = b$, but this leads to clustering of points around the poles $\theta = 0$ and $\theta = \pi$, see Figure 3. In other words, this method fails to generate uniformly distributed points on a sphere of radius r=b.

A better idea would be to use the Gaussian distribution, since it's spherically symmetrical. In Figure 3 one can observe that this method indeed seem to be working.



(a) (b)

Figure 3: (a): 3000 points (x, y, z) generated on a sphere of radius 1 using the naive method (b): 3000 points (x, y, z) generated on a sphere of radius 1 using a Gaussian distribution

2.4 Self-avoiding freely jointed chain walk

This random walk was implemented by using the following algorithm:

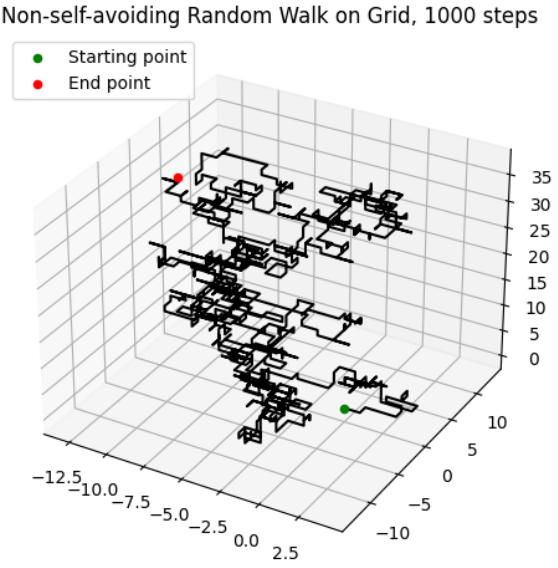
- Initiate a walk by creating a positional vector $\mathbf{r} \in \mathbb{R}^3$ in the origin.
- Generate a vector $\mathbf{b} \in \mathbb{R}^3$ of length b with a random orientation using the Gaussian distribution.
- Construct a vector $\mathbf{r}_{temp} = \mathbf{r} + \mathbf{b}$
- Check if $\mathbf{r}_{temp}$ is more than a distance d from all the previously visited positions in the walk. If this is the case continue to the next item in this algorithm, otherwise go back to item 2.
- Take a step $\mathbf{b}$ by setting $\mathbf{r}$=$\mathbf{r}_{temp}$

It's worth noting that $d \leq b$ in order for this method to work.

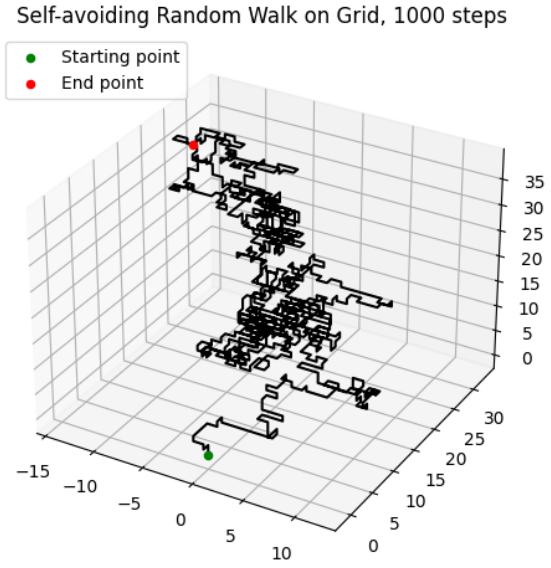
2.5 Simulation

Figure 4 shows simulations of random walks on a grid. Figure 5 shows simulations of freely jointed chain walks. To simplify things, the step-length b and the distance d (which is used in the self-avoiding freely jointed chain walk) are both set to be equal to one.

2.5.1 3D random walk on grid



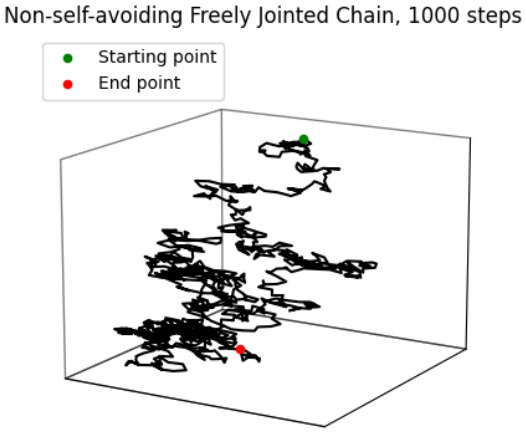
(a)



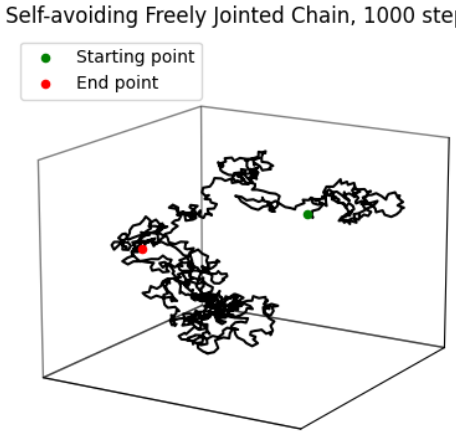
(b)

Figure 4: (a): Non-self-avoiding (b): Self-avoiding

2.5.2 Freely jointed chain walk



(a)



(b)

Figure 5: (a): Non-self-avoiding (b): Self-avoiding

3 Analysis

For each method, we want to analyze the end-to-end vector $\mathbf{R}$. Since random walks are stochastic processes we need to analyze the statistical average over many walks. In the non-self-avoiding cases we have that

$$< \mathbf{R} >=< \sum_{i=1}^{n} \mathbf{b}_i >= \sum_{i=1}^{n} < \mathbf{b}_i >= \mathbf{0}$$

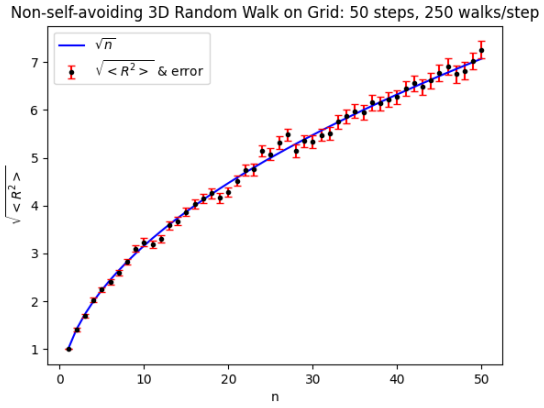
since all directions are equally probable. Instead we need to analyse the root mean square distance, which in the non-self-avoiding case can be shown[2] to be

$$\sqrt{< R^2 >} = b\sqrt{n} \tag{1}$$

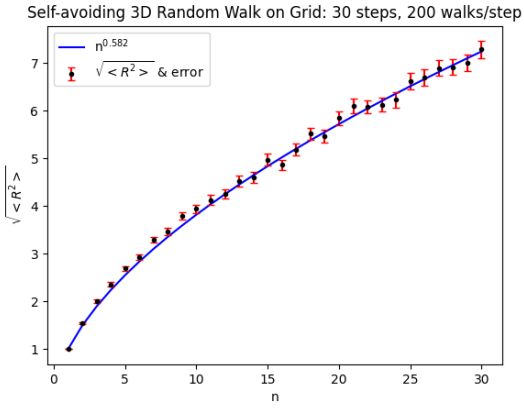
where b, as mentioned before, is the step-length and n is the number of steps taken. To analyze the error the variance, which is given by $Var =< R^2 > - < R >^2$, is used. The standard error estimate is calculated as $SEE = \sqrt{Var\frac{1}{N-1}}$ where N is the number of walks and the $\sqrt{< R^2 >} - fluctuation$ is calculated as $\sqrt{Var\frac{N}{N-1}}$.

As mentioned before, b and d are both set to one.

3.1 3D random walk on grid

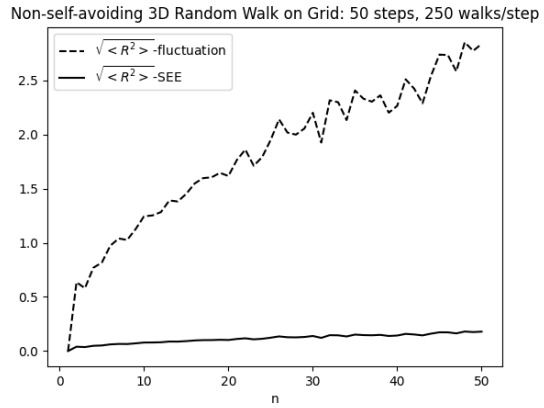


(a)

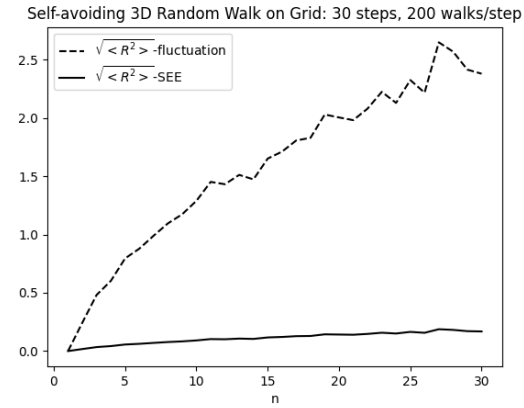


(b)

Figure 6: $\sqrt{< R^2 >}$ plotted against number of steps n. (a): Non-self-avoiding walk on a grid (b): Self-avoiding walk on a grid



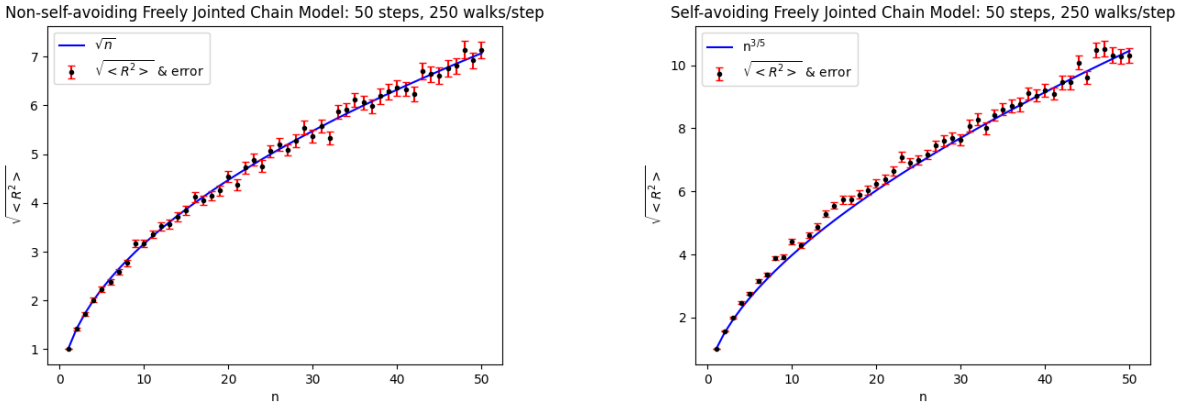
(a)



(b)

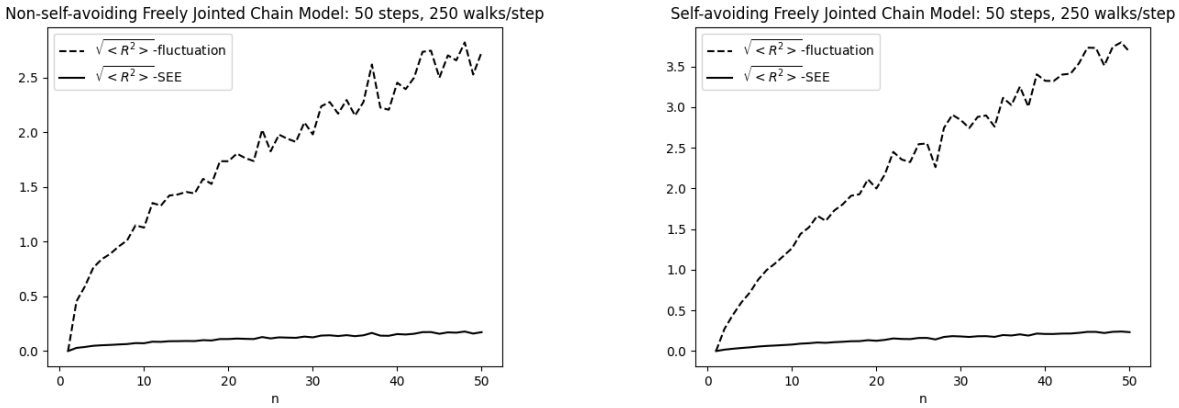
Figure 7: SEE and $\sqrt{< R^2 >}$-fluctuation plotted against number of steps n. (a): Non-self-avoiding freely jointed chain walk (b): Self-avoiding freely jointed chain walk

3.2 Freely jointed chain walk



(a) (b)

Figure 8: $\sqrt{< R^2 >}$ plotted against number of steps n. (a): Non-self-avoiding freely jointed chain walk (b): Self-avoiding freely jointed chain walk



(a) (b)

Figure 9: SEE and $\sqrt{< R^2 >}$-fluctuation plotted against number of steps, n. (a): Non-self-avoiding freely jointed chain walk (b): Self-avoiding freely jointed chain walk

3.3 Comparison

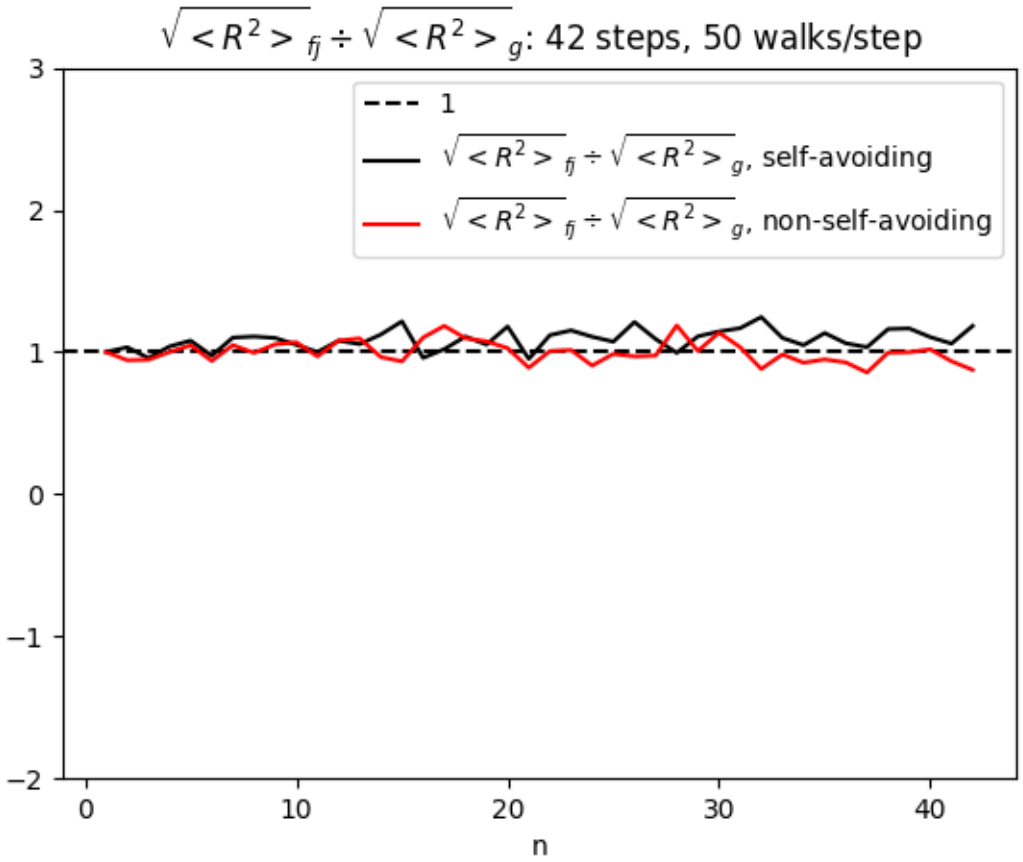


Figure 10: Fraction of $\sqrt{< R^2 >}_{fj}$, the RMS-end-to-end distance for the freely jointed chain walk, and $\sqrt{< R^2 >}_g$, the RMS-end-to-end distance for the random walk on a grid, plotted against number of steps n.

4 Discussion and conclusions

$\sqrt{< R^2 >}$ roughly grows as $\sqrt{n}$ in the non-self-avoiding 3d random walk on a grid and roughly goes as $n^{0.582}$ in the self-avoiding case which makes sense since one could expect the end-to-end distance to grow a bit faster in the non-self-avoiding case. See Figure 6.

$\sqrt{< R^2 >}$ roughly grows as $\sqrt{n}$ in the non-self-avoiding freely jointed chain walk, which corresponds well with Equation 1, and just as in the 3d random walk on a grid it grows faster, approximately as $n^{3/5}$, in the self-avoiding-case. See Figure 8.

It's slightly odd that some data points in Figure 6 and 8 does not, at least within the margin of error, behave like $\sqrt{n}$. This may be due to rounding errors.

The self-avoiding walk on a grid tends to get stuck in a loop after, on average, roughly 3000 steps (although the variance of this quantity is high). This problem does not appear for the self-avoiding freely jointed chain walk.

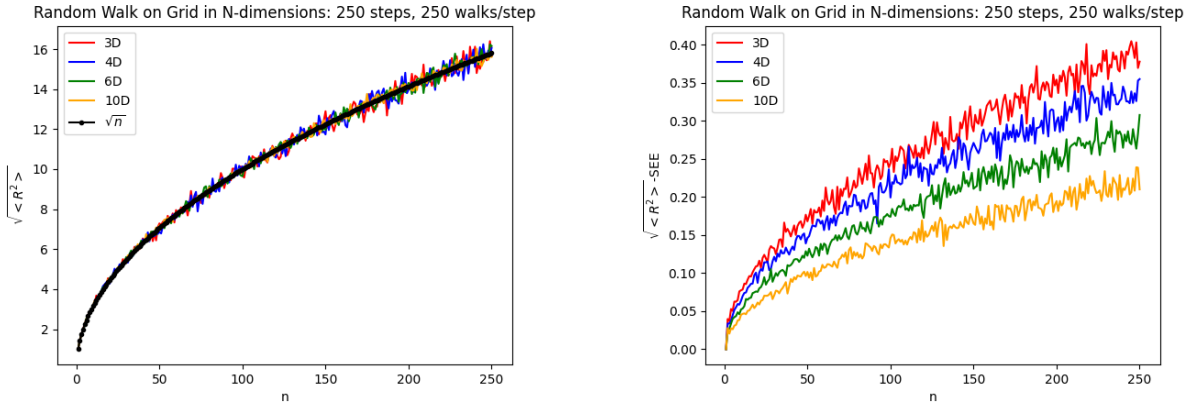
In Figure 7 and 9 one can observe that the SEE and $\sqrt{< R^2 >}$-fluctuation increases when n increases.

The asymptotic behavior of the $\sqrt{< R^2 >}$ for the different walks seem to be approximately the same in the case when b=d=1 which can be observed in Figure 10. The behavior of the self-avoiding freely jointed chain walk roughly varies between $\sqrt{n}$ and $n^{3/5}$ as d varies between 0 and b. It's worth mentioning that when d gets closer to zero $\sqrt{< R^2 >}$ tends to $\sqrt{n}$, which what one would expect.

If one wants to simulate many steps, for example if one wants to simulate an ideal polymer with a large amount of monomers, the self-avoiding freely jointed chain walk seem to be a better option than the self-avoiding walk on a grid since it rarely, if ever, "get stuck" in a loop.

5 Project extension: Non-self-avoiding random walk in N dimensions

Since me and Melker Wallén have collaborated on this project we decided to extend it by studying non-self-avoiding random walks in an arbitrary number of dimensions.



(a) (b)

Figure 11

Somewhat surprisingly, as can be seen in Figure 11, the $\sqrt{n}$-dependence seem to be independent from the number of dimensions and the SEE seem to be decreasing as the number of dimensions increases. It might be worth pointing out that the challenging part of this project extension was being able to write a program that simulates random walks on a grid in an arbitrary number of dimensions. The code can be found in the appendix.

6 Appendix A: Code listings

6.1 Non-self-avoiding ND random walk on grid

```
import numpy as np

def direction_picker(dim):
    return np.random.randint(0, 2*dim)


def randwalk(n, dim):

""" Random walk of n steps on a grid in dim dimensions.
Returns end-to-end-distance and end-to-end-distance-squared """

    positions = np.zeros((n+1, dim))

    for i in range(1, n+1):
        positions[i] = positions[i-1]
        val = direction_picker(dim)
        if val % 2 == 0:
            change = 1
        else:
            change = -1
        positions[i][val//2] = positions[i-1][val//2] + change
    R = np.linalg.norm(positions[n])
    return R, R**2
```

References

[1] *Random Walk*. 10 January 2021, at 14:17 (UTC). URL: https://en.wikipedia.org/wiki/Random_walk (visited on 01/17/2020).

[2] *Ideal chain*. 24 December 2020, at 00:23 (UTC). URL: https://en.wikipedia.org/wiki/Ideal_chain (visited on 01/17/2020).