A simulation of polymer growth

Miranda Naaktgeboren 4569644
Luuk Zonneveld 4563256

TU Delft,
Faculty of Applied Sciences,
MSc Program Applied Physics

Delft, May 19, 2020

Abstract

This report contains the documentation of a self avoiding random walk simulation. To create the self avoiding random walk polymers are initialised and grown up to size $N = 250$. For every step weights are calculated, based on the Lennard-Jones potential, that give the possibility of new bead placements. According to a Monte Carlo method the new positions are chosen. This algorithm is called the Rosenbluth algorithm. Additionally another algorithm called PERM (pruning and enriching Rosenbluth method) has been implemented and tested. To test the algorithms, the end-to-end distance of polymers is calculated. Combined with error calculation the results are compared to the expected behaviour of the length as function of polymer length.
The results show that PERM is outperformed by Rosenbluth which goes against existing literature. The Rosenbluth algorithm is considerably faster and produces more use able polymers compared to PERM. The difference in performance can be caused by a sub-optimal performance of PERM. To draw further conclusions the PERM algorithm needs to be optimised further.

Contents

Abstract **i**

1 Introduction **1**

2 Methods **2**
2.1 Rosenbluth 2
2.2 Weights and probability 2
2.3 PERM 4
2.4 Error 4
2.5 Validation 5

3 Results **6**
3.1 Self avoiding random walk 6
3.2 Random walk 8

4 Discussion **10**
4.1 Self avoiding random walk 10
4.2 Random walk 10
4.3 Performance 10

5 Conclusion **12**

List of References **13**

Appendix A **14**

1 Introduction

In order to better understand how polymers work one important aspect is to understand how they grow. For this simulations can be run to see how the polymers grow and curl up in ideal conditions. In this project the growth of a polymer has been simulated. Two different algorithms are used to achieve this end goal. First the Rosenbluth algorithm and then the PERM algorithm, which stands for pruned and enriched Rosenbluth method. This project is based on the project by Jos Thijssen [1].
Polymers can be simulated in different ways, in this report it is an off lattice simulation. Furthermore the polymers are simulated as if they were to grow in a good solvent, also known as a dilute polymer solution. The beads are repelling each other which results in a self avoiding random walk, for which the interaction can be calculated with the Lennard-Jones potential. Another possibility is that there is no interaction between the beads, then it is called a random walk or the ideal chain case.
One important aspect of the polymer growth to look at is the end-to-end distance. The end-to-end distance does depend differently on the number of beads when there is interaction compared to when there is no interaction. In section 2 the methods used in the simulations is described with the corresponding theory behind it. In section 3 the results are given. In section 4 the results are discussed as well as the performance of the code and in section 5 a conclusion is drawn.

2 Methods

2.1 Rosenbluth

The simulation of the polymers is started with two beads which are one unit length apart from each other. Then the polymer is extended by adding an extra beat one by one. Where this new beat is placed, is based on the weights assigned to every possibility. The weights will be explained more extensively in the next section 2.2. The weights are based on the Lennard-Jones potential which is given by equation 1. By taking into account the interaction a self avoiding random walk is created.

$$U = 4\epsilon\left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6}\right] \tag{1}$$

In this formula $\epsilon = 0.25$ and $\sigma = 0.8$ the same as used in the literature [1]. Furthermore r is the distance between the beads and U is the potential energy.
This algorithm is used and repeated to get a set of polymers that are all created from a random distribution. Section 2.2 explains how it is possible to get this distribution and how the polymers will be filtered to get adequate results.

2.2 Weights and probability

For each new bead the possibility of the bead being placed there is saved as the weight, w_i. This weight depends on the angle θ_i as can be seen in equation 2. In this simulation it has been taken that $T = 1$ and $k_B = 1$. The total probability of where the new bead can be places is given by W, which can be calculated as in equation 3.

$$w_i = exp(-U(\theta_i)/(k_B T)) \tag{2}$$

$$W = \sum_i w_i \tag{3}$$

In this simulation for each new bead there are six random angles θ_i where it can be placed on a distance of unit length 1. The angles are determined by taking 0 to 2π and diving that into steps of $\frac{2\pi}{6}$ then adding a random number between 0 and π.
The probability of accepting a certain angle θ_i is determined by $\frac{w_i}{W}$. This is done by performing a Monte Carlo process that relies on generating a random number between 0 and 1. This random number must then be related to the probabilities of each angle θ_i. Because the fraction $\frac{w_i}{W}$ will always be bounded by lower bound 0 and higher bound 1 it is possible to use these values in combination with the randomly generated number. An example of six weights is shown in figure 1. If you would generate a random number with the value 0.5 the corresponding angle would be θ_3 as seen in figure 1, but would the random number be 0.75 the result will be θ_5.

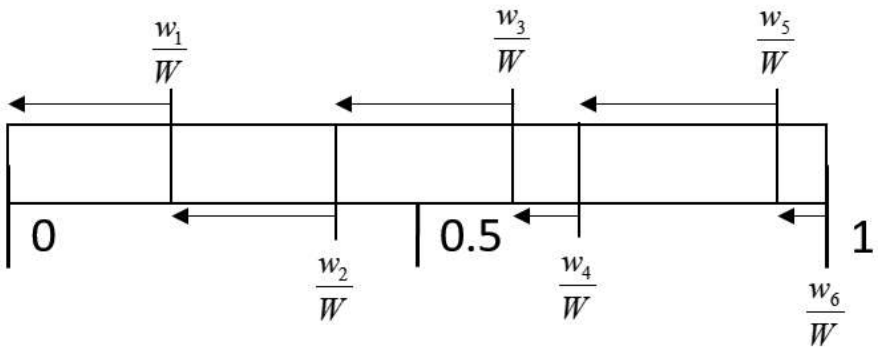


Figure 1: Distribution of the fractions over the domain from 0 to 1. The arrows indicate which part of the domain corresponds to which indices of the weight w_i and θ_i.

To find out if certain polymers have a high probability of existing it is possible to look a the weight W. If the value of the weight is very small it means that none of the new possible angles give a probable growth of the polymer. To quantify the possibility of the polymer that has grown to size N it is possible to multiply all the weights W at every step to get the probability P_n at polymer length n.

$$P_n = \Pi_{i=3}^{n} W_i \tag{4}$$

Here it is assumed that probabilities start to play a role when the first bead is added so i starts at 3.
In some cases the weight W is so small that no domain can be determined by the algorithm. To make sure that the system still continues it was decided to choose the value of the last angle at such instances. It should be noted that this gives a bias to the system but it is expected that it will have little to no influence on the results. This assumption rises from the method that is used to average the lengths of simulated polymers. In this case the observable of the system is the squared end-to-end distance, R_e^2. The average weighted end-to-end distance is calculated, using the following formula.

$$\langle A \rangle = \frac{\sum_i W_i A_i}{\sum_i W_i} \tag{5}$$

Because formula 5 directly uses the weight W, the polymers which had improbable steps in the system will effectively have no effect on the observable $\langle A \rangle$.
Because the improbable polymers will still make up part of the simulation it is not an effective way to generate them. In a attempt to counter this, another algorithm is tested that should reduce the amount of improbable polymers. This algorithm is called PERM and discussed in section 2.3

2.3 PERM

PERM stands for pruning and enriching Rosenbluth method, which means that the strong polymers parts are repeated and the weak polymer parts are removed. As a result of this the fully grown polymers, in this case up to 250 beads, should have a higher probability to occur than with the Rosenbluth algorithm.
The basic idea behind this algorithm is to duplicate the polymers that have a high probability P and remove any polymers that have a low probability at any point in the simulation. The starting point of PERM can be chosen depending on the polymer length and will be discussed more in section 3. The limits for removal and duplication are given by the literature [1]:

$$\begin{aligned} Uplim &= 2\bar{W}/W_3 \\ Lowlim &= 1.2\bar{W}/W_3 \end{aligned} \tag{6}$$

Here $\bar{W}$ is the average weight of a polymer at the every step of the algorithm and W_3 the weight when the first bead is added.
To make sure that the probability of the polymers does not change the outcome of the distribution the weights of replicated polymers will be halved together with the weight of the polymer that is created. If a polymer has a low probability of existing it gets removed with a chance of $\frac{1}{2}$. If the polymer is not removed the weights are doubled to make sure that the distribution stays the same in the end.
This process should make sure that the polymers that have a higher probability of existing according to the theory, are more probable to emerge from the simulations. The direct result should be to reduce the amount of computed polymers to get results with uncertainties that are acceptable. The errors that are calculated in both the Rosenbluth and PERM algorithm are explained in section 2.4.

2.4 Error

Since the data is generated randomly it is valid to use an error calculation where the results are statistically non-correlated from each other. Equation 7 shows the error calculation of non-correlated weighted data [2]. The weights are used in the calculation of the observable A, which was already shown in equation 5. As a result of this the error does depend on the weights.

$$\sigma_A[N] = \frac{\sqrt{\frac{1}{n-1} * \sum_i (A_i - \langle A \rangle)^2}}{\sqrt{n}} \tag{7}$$

With N the number of beads, A the observable and n the number of non-zero elements in the weight. It should be noted that the data in PERM is not completely statistically independent. However, it can still be approximated as statistically independent.

2.5 Validation

For the validation the end-to-end distance, R_e, is calculated and compared with the known literature. The end-to-end distance is the distance between the first and last beat of the polymer. For this multiple polymers of the same number of beads are grown and the weighted average is calculated. To compare it to the literature the R_e^2 will be plotted as a function of N [1]. It should be fitted as $R_e^2 = a * (N-1)^{2\nu}$, with a the simulation specific constant and $\nu = 0.75$ for a self avoiding random walk in two dimensions.

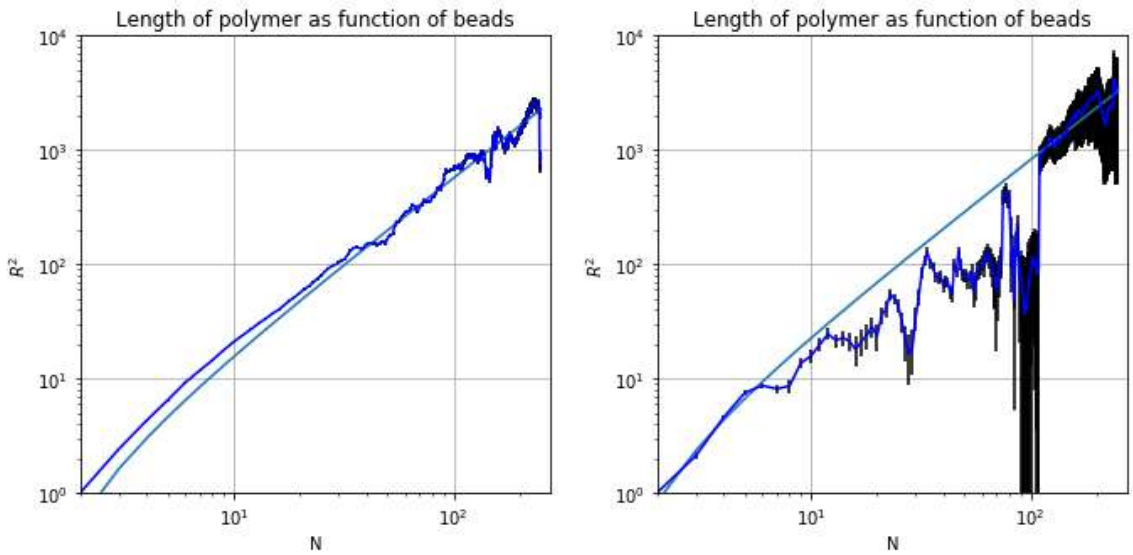
Another validation of the code is done by taking into account the random walk or ideal chain case, where there is no bead interaction. If there is no interaction, all the weights of the place for the new bead are equal, which results in $R_e \propto N^{\frac{1}{2}}$ [3]. So for the random walk the data will be fitted as follows $R_e^2 = a_r * N$ with a_r a constant. In the results the end-to-end distance will be given as a function of the number of beads.

3 Results

In this section the results of the algorithms are discussed. This is done by showing the simulation of polymers with interaction between the atoms and by simulating without interaction effectively making it a random walk.

3.1 Self avoiding random walk

The algorithm has been tested for a few different parameters. For the Rosenbluth algorithm this means that the iterations have been varied. For PERM the iterations have also been varied but one additional parameter, that determines from which value of N the pruning and enriching comes into play has been altered.
Every result is fitted to a function $\alpha(N-1)^{1.5}$ which represents the expected behaviour. The fit contains no error bars and is colored light blue for every simulation. The data points are shown with error bars that are black.
First the algorithms were compared for both 1000 iterations and PERM directly being used from polymer length 3. The results are shown in figure 2.



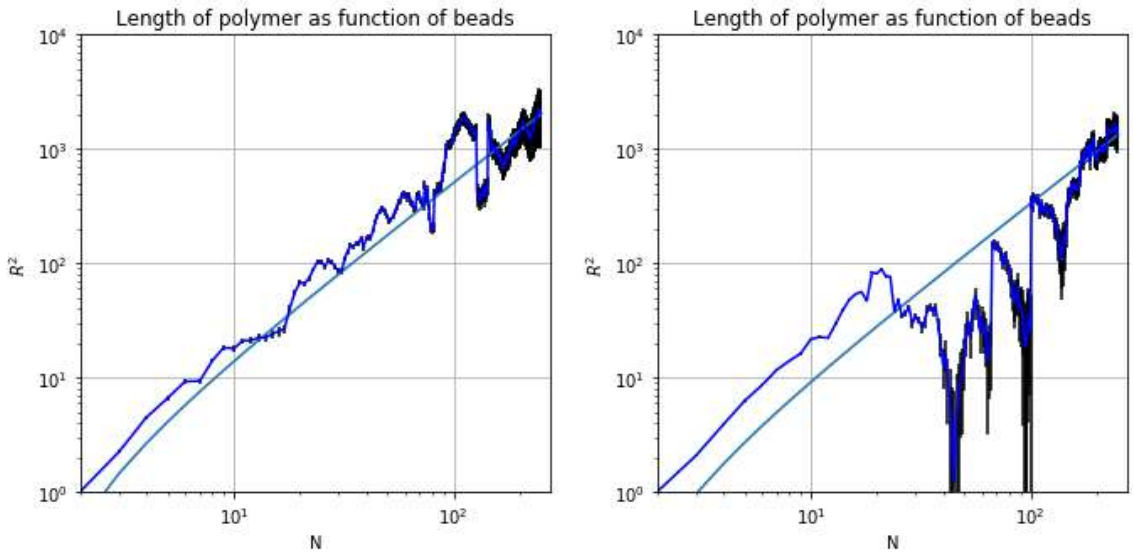
(a) Results from 1000 polymers created with Rosenbluth algorithm

(b) Results from 48 polymers created with PERM by running the algorithm 1000 times.

Figure 2

It can directly be seen in figure 2 that the Rosenbluth algorithm outperformed the PERM method. The difference lies in the amount of polymers created. Where for PERM only 48 polymers were used there were 1000 polymers for the Rosenbluth method. This could be

explained by starting to remove polymers when the length is only 3 beads long.
To make sure that more polymers would be usable the PERM algorithm was started from length $N = 3$, 70 and 100 to test how many would be created and how good the results would be, as image 2b shows bad results. As already can be seen from the Rosenbluth method the deviations only start to occur after length $N = 10$.

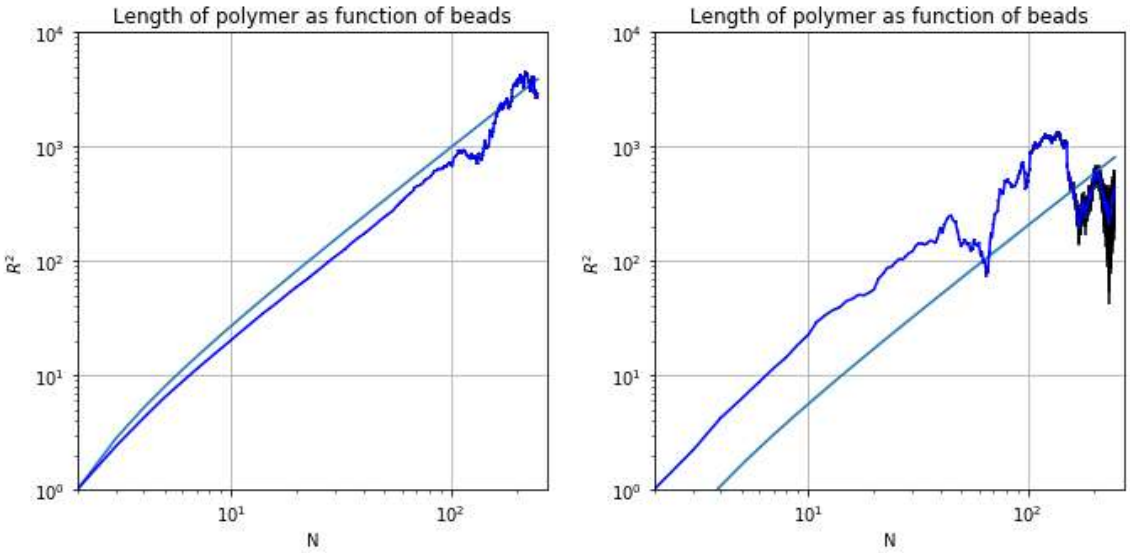


(a) Results from 209 polymers created with PERM algorithm run for 1000 iterations.PERM was started at length $N = 70$.

(b) Results from 913 polymers created with PERM by running the algorithm 1000 times. PERM was started at length $N = 100$.

Figure 3

Figure 3 shows that starting PERM from length $N = 70$ seems to work best compared to starting at $N = 3$ or $N = 100$. The removal and multiplication of polymers seems to be on time and for lower values $N < 70$ there seem to be comparable fluctuations as starting when $N = 3$.
Keeping in mind that the start value with the best results for now is $N = 70$ it was decided to perform both Rosenbluth and PERM for 10000 iterations to see which algorithm is best.



(a) Weighted average of 10000 polymers created with the Rosenbluth algorithm.

(b) Weighted average of 1997 polymers created after 10000 iterations with the PERM algorithm.

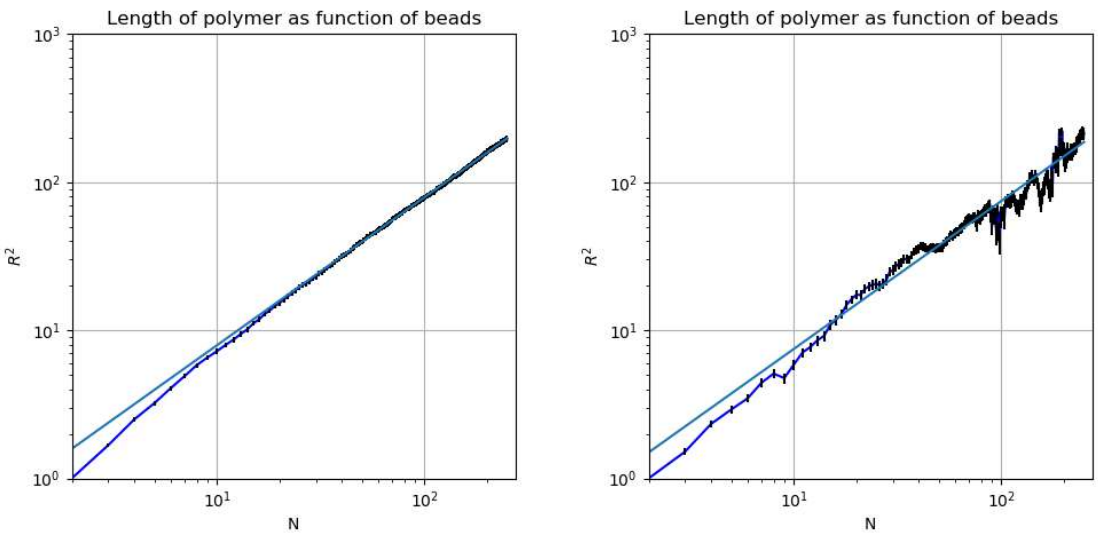
Figure 4

The results seen in figure 4 are not corresponding to what is shown in the literature [1]. It should be the case that the simulations behave as $\alpha(N-1)^{1.5}$. The fit of the Rosenbluth algorithm in figure 4a does show some correspondence to this fit, but the data points are a bit lower than the fitted line. The deviation that starts around $N = 100$ is in correspondence with what was shown in the literature. However it was expected that the with the PERM algorithm the results would as good as in Rosenbluth and then show an improvement for $N > 100$ and this is clearly not the case. Though, it should be noted that in general the data points in figure 4b do follow the same trend as the fit. The fit is a lot lower than the data points because at the end the data points suddenly drop and so lowering the fitting constant.
Because the simulations do not show the behaviour that is expected according to [1] another test has been performed. In the following section the results of a more general test are shown, namely the random walk.

3.2 Random walk

In figure 5a the weighted fit of a random walk using the Rosenbluth algorithm is shown. The fitted line is given by the light blue line and the data points by the dark blue line with black error bars. The simulation was run for 1000 iterations and as can be seen this results in a line which can be fitted for $R_e^2 = a_r * N$ and has small error bars. In figure 5b the results are shown of the same test with the PERM algorithm. For this simulation

1000 iterations were used, but this resulted in only 242 polymers. The PERM algorithm was activated from polymer length $N = 70$, since that seemed to be the more favourable value compared to $N = 3$ and $N = 100$.



(a) The random walk with the Rosenbluth algorithm and a fitted line. The fitted line has the constant $a_r = 0.792$ and $\pm 2.83e - 07$.

(b) The random walk with the PERM algorithm and a fitted line. With the constant for the fitted line being $a_r = 0.748$ and $\pm 5.67e - 05$..

Figure 5

The results are not as was expected, because the expectancy was that the PERM algorithm would give a more stable result than the Rosenbluth algorithm. The first difference is the number of polymers that were used in the end. Because the 1000 iterations resulted in only 242 useful polymers in the PERM algorithm. This big difference in number of polymers can explain why Rosenbluth is more stable. However the most important part of the the random walk is that both the Rosenbluth and PERM algorithm can be fitted to the expected behaviour [3], which shows that for the random walk both algorithms seem to work.

4 Discussion

4.1 Self avoiding random walk

The behaviour of the Rosenbluth and PERM algorithm are shown in figure 4. According to the literature [1] the results of PERM should be better than Rosenbluth but the findings are reversed. There is no direct indication what is going wrong. Even changing the value at which bead the pruning of PERM starts has not yielded a clear optimum as the best value that was found ($N = 70$) did not yield better results than Rosenbluth. It could be beneficial to test PERM for more starting values to find an optimum. This was not done as the algorithm took a lot of computation time.
The speed of the algorithms are also an important aspect in general and are discussed in more detail in section 4.3. The general gist is however that Rosenbluth is way faster than PERM.

4.2 Random walk

The main difference in the random walk is that the Rosenbluth is more stable than the PERM algorithm, but on the other hand the number of used polymers was almost 4 times lower in the PERM algorithm. This result is not what was expected, because the PERM algorithm should be an improvement of the Rosenbluth algorithm. On the other hand the results are difficult to compare, because if the simulation would be run again, there would be a different result with maybe a lot more stable polymers, which could give a more stable graph for PERM. The more important part is that the graph can be fitted to the expected behaviour, which shows that the simulation does work as expected even tough not being as stable as expected in the case of PERM.
Combining this with the results of the self avoiding random walk it seems that in general the simulation works, but the main problem with PERM is that there are not enough polymers to have a good weighted fit. This problem could not be solved, since the running time of the PERM algorithm is very long. This will be discussed in the next section 4.3.

4.3 Performance

An important aspect of the algorithms is of course the performance. If the Rosenbluth would be outperformed by the PERM algorithm it would not immediately mean that it would be a better option to use PERM.
In the coded algorithms it was encountered that PERM has a relative slow performance compared to the Rosenbluth algorithm. It is expected that this originates from the method by which polymers are copied and removed. In table 1 the results are shown for 10000 iterations of both the Rosenbluth and the PERM algorithm.

Table 1: Comparison between the PERM and Rosenbluth algorithm. Both ran for 10000 iterations.

Method	Rosenbluth	PERM
Polymers	10000	1997
Time [min]	14.7	132

From table 1 it can be seen that the PERM algorithm costs significantly more time to run the same amount of iterations. This gives the possibility to just run the Rosenbluth algorithm more than the PERM algorithm and get better results.
Estimating how long it would take to get the same amount of sample polymers for PERM would mean that it would take around 50000 iterations and 660 minutes to get 10000 polymers. This result limits the use of PERM and gives a big advantage to Rosenbluth.
It is however not the case that the performance of the PERM algorithm is the best that could be achieved. If more time were to be invested in optimisation it is expected that PERM could be programmed to operate faster. However, it is inherent of the PERM algorithm that more computations are performed so it is also expected to be slower than Rosenbluth although not as much as is observed in the project.
If the results of the PERM algorithm would have been significantly better than for Rosenbluth then it might not even be an issue if way less iterations are needed and more polymers are simulated that have a relative high probability. But in this simulation it was clearly not the case that the PERM algorithm gave better results despite having less iterations, which means that the long computation time of the PERM algorithm was a problem in generating good results for the PERM algorithm.

5 Conclusion

In this report the Rosenbluth algorithm outperforms the PERM algorithm, both in higher quality results and faster simulation times. This is not what was expected from the literature [1]. In section 3.2 it is shown that the algorithm does perform as expected without the Lennard-Jones interaction. This leads to believe that the main difference between Rosenbluth and PERM originate from different amount of polymers generated. This in term directly depends on the starting value at which the pruning starts. Starting early should give more highly probable polymers but also less in quantity. Starting early does however give some complications as the PERM algorithm was not optimised and performed substandard compared Rosenbluth. For the same amount of iterations PERM took approximately nine times longer and produced only a fifth of the polymers compared to Rosenbluth.
PERM should not be written off as a valid option as there are doubts if the implementation has a sufficient quality. To be able to conclude which performance would be better the PERM algorithm should first be checked and optimised to be able the make an equal comparison.

References

[1] Jos Thijssen. *Computational physics.* Cambridge university press, 2007.

[2] Ifan Hughes and Thomas Hase. *Measurements and their uncertainties: a practical guide to modern error analysis.* Oxford University Press, 2010.

[3] David P Landau and Kurt Binder. *A guide to Monte Carlo simulations in statistical physics.* Cambridge university press, 2014.