Monte Carlo simulations of polymers

AP3082-CP

Delft University of Technology

Isacco Gobbi (5161347), Ludwig Hendl (5154375)

19th May 2020

Abstract

A two dimensional polymer growth in a good solvent is simulated on the basis of the Rosenbluth algorithm and the PERM algorithm[1]. Accurate results are reproduced and compared to literature. The observables end-to-end distance as well as radius of gyration exhibit the expected behavior and allow insight in the growth of polymers by random walks (RW) and self avoiding random walks (SAW). Advantages of the PERM algorithm are shown within this report.

Contents

1 Introduction

Polymers and their behaviour is a matter of interest for many professionals working academically or as a part of the industry dealing with all sorts of long chemically bonded chains. Polymers became available throughout the 19^{th} century which laid the foundation for the study of their nature. The physics underlying these compounds is in general very complex but a surprising amount of insight can be derived from the simplest models, studied as in the field of statistical mechanics.

In this work we consider polymers in a good solvent meaning that the free energy of a polymer piece surrounded by the solvent is lower than the energy of a piece close to another part of the polymer.[3] In this approximation each polymer is effectively independent of the others and their interactions can be neglected. A model where a polymer is represented as a chain of beads (monomer) consecutively bonded to the previous and subsequent beads with a predefined and fixed distance can be used as an idealization for basic polymer behavior. The interaction of beads can be modelled by a Lennard-Jones potential as a weak, long-range attraction combined with strong short-range repulsion between monomers. The resulting polymer chains lead to insights on the macroscopic properties of such a system, in particular how such structures grow as a function of the number of monomers.

Expectation values are obtained by the means of the Monte Carlo method in an ensemble of two-dimensional polymers without an underlying lattice such that no bead position is predefined.

2 Method

2.1 Monte Carlo method

The algorithm is based on the Monte Carlo (MC) method, which is a useful approach to compute multidimensional integrals based on the random sampling of a known probability distribution.

Like all integration algorithms Monte Carlo methods are based on the discretization of an integral into a sum:

$$I = \int_a^b dx f(x) \simeq \frac{(b-a)}{N} \sum_{i=1}^{N} w_i f\left(x_i\right) \tag{1}$$

where w_i are the weights associated with the mesh. The difference between Monte Carlo and other integration algorithms is that the mesh points x_i are *randomly distributed* in the (a, b) interval and all weights are equal to one.

In many problems in computational physics this is particularly appreciated due to the high dimensionality of the configuration space, since it would require an unfeasible amount of time to integrate over it with more naive integration algorithms.

A typical example of this is the calculation of the *expectation value* of a physical quantity A. In the canonical ensemble this is defined as:

$$\langle A \rangle = \frac{1}{Z} \int A(q,p) \mathrm{e}^{-\beta H(q,p)} dq dp \tag{2}$$

where Z is the *partition function*.

By the central limit theorem $\langle A \rangle$ can be estimated by its *mean value*:

$$\langle A \rangle \simeq \overline{A} = \frac{1}{N} \sum_{i=1}^{N} A(X_i) \tag{3}$$

where $X_i = (p_i, q_i)$ are *randomly distributed* points in the configuration space. The same theorem also provides the *standard error of the mean* which scales as $\frac{1}{\sqrt{N}}$.

Nevertheless, blindly sampling the configuration space will give poor results of the mean value: the distribution of Boltzmann weights $e^{-\beta H}$ is sharply peaked around its expectation value at equilibrium, hence only few sampling points will give the greatest contribution to the integral, leading to great statistical error. One technique to reduce this is called *importance sampling* and consists in taking a weighted average of the samples. The weights are calculated according to a known probability density function $P(p, q)$ chosen to resemble the Boltzmann distribution:

$$\begin{aligned}\langle A \rangle &= \frac{1}{Z} \int A(X) \mathrm{e}^{-\beta H(X)} dX = \frac{1}{Z} \int \frac{\mathrm{e}^{-\beta H(X)}}{P(X)} P(X) A(X) dX \\ &= \frac{\int P(X) W(X) A(X) dX}{\int P(X) W(X) dX} \simeq \frac{\sum_i W(X_i) A(X_i)}{\sum_i W(X_i)}\end{aligned}$$

the weights take the form $W(X) = \frac{e^{\beta H(X)}}{P(X)}$ and X_i are distributed according to $P(X)$.

2.2 Rosenbluth algorithm

One clever way to generate an ensemble of polymers was proposed by Rosenbluth and Rosenbluth in 1955. Here we present a different version, adapted for off-lattice polymers. Each polymer starts as a monomer, then the third bead is attached at an angle to the second in the following way:

- Choose a random offset angle

- Choose N_{res} equally spaced positions on the circle starting from the offset angle
- Compute the Boltzmann weight of each trial position j: $w_j = e^{-\beta H}$
- Draw a random number between 0 and $W = \sum_j w_j$
- Choose angle $\tilde{j}$ corresponding to drawn number
- Repeat for all remaining beads

This way position j is drawn with probability $\frac{w_j}{W}$. The total weight for a polymer is therefore:

$$\prod_{l=3}^{N} \frac{w_{\tilde{j}}^{(l)}}{W^{(l)}}$$

N is the number of beads in the polymer and $\tilde{j}$ addresses the chosen position for every bead. Each polymer is then generated independently from the others in the ensemble.

In order to compute the mean value for any physical quantity of this population a remark must be made. The actual Boltzmann distribution we wish to sample has the form:

$$e^{-E_{\text{total}}/(k_b T)} = \prod_{l=3}^{N} w_{\tilde{j}}^{(l)} \tag{4}$$

Therefore we must weigh each configuration by its weight $W_{poly} = \prod_{l=3}^{N} W^{(l)}$ when computing the ensemble average:

$$\overline{A} = \frac{\sum_{poly} W_{poly} A_{poly}}{\sum_{poly} W_{poly}} \tag{5}$$

The error of this weighted average can be calculated by:

$$\sigma_{\overline{A}}^2 = \frac{N}{(N-1)\sum_{poly}^{N} W_{poly}^2} \sum_{poly}^{N} W_{poly}^2 \left(A_{poly} - \overline{A}\right)^2 \tag{6}$$

where N is the number of elements over which the weighted average was calculated.

2.3 PERM

The limits of Rosenbluth algorithm emerge when generating ensembles of longer polymer chains: the statistical errors grow large with larger numbers of beads. It is easy to notice that as the polymer length grows it becomes less and less likely to produce polymers with high weights. This is due to the fact that the Boltzmann distribution of the polymers becomes exponentially large with increasing N and is therefore harder to sample 'good polymers'. One attempt to overcome this problem is the 'Pruned-Enriched Rosenbluth Method' (PERM). It consists in dynamically generating the polymer population by 'enrichment' of good samples and 'pruning' of bad samples. A good polymer is enriched by adding the a copy of it to the population and dividing their weights by half (so that the total population weight stays the same). On the other hand a bad sample is pruned (removed) with 50% probability. If it survives the pruning, its weight is doubled.

This way the population grows maintaining higher number of high-weight samples. The criteria for enrichment and pruning are based on the average weight of the population $\overline{W}$, e.g. a polymer is enriched only if

$$W_{poly} > \alpha_{up}\overline{W} \tag{7}$$

The same holds for pruning but the direction of the inequality is reversed. α_{up} and α_{low} determine whether the population will grow, shrink or remain stationary and in general can depend on the bead number.

Originally PERM was implemented with a recursive procedure which allows to store only one polymer in memory at time during the computation. To copy a polymer the recursive function is called twice and the existing polymer effectively 'splits' into two subcopies that are then grown independently. This approach is called depth-first and allows for very little memory usage. The downside for this kind of approach is the difficulty in setting up the simulation parameters, as the recursive method doesn't allow to monitor the simulation step-by-step, leading to a very hard code to debug.

In this project we opted for a breadth-first version of PERM: an initial population of monomers is generated and all polymers are grown in parallel one bead at the time. At every timestep the average population weight is computed and good samples are enriched while bad ones are pruned. This method, despite being more memory intensive than the precedent one, benefits of better usability and transparency.

2.4 Physical quantities

In this project physical quantities are extracted and averaged from a set of bead by bead grown polymers. The focus lies on two observables: end-to-end distance S and radius of gyration R_g.

- *End to end distance squared* S^2:

$$S^2 = \left\langle (\mathbf{R}_N - \mathbf{R}_1)^2 \right\rangle \tag{8}$$

Here $\mathbf{R}_N$ is the N^{th} and $\mathbf{R}_1$ the first bead position vector. The end-to-end distance squared S^2 is equal to R_N^2 if the first bead is placed at the origin. The end-to-end distance is expected to scale with the number of beads N as:[3]

$$R_N^2 \propto N^\nu \tag{9}$$

This quantity is particularly interesting since it directly compares the behavior of polymers with a random-walk or self avoiding random walk (SAW). This will be addressed in further detail in the following section.

- *Radius of gyration squared R_g^2:*

$$R_\mathrm{g}^2 = \frac{1}{N}\sum_{i=1}^{N}\left\langle(\mathbf{R}_i - \mathbf{R}_\mathrm{CM})^2\right\rangle \tag{10}$$

where $\mathbf{R}_\mathrm{CM} = \frac{1}{N}\sum_{i=1}^{N}\mathbf{R}_i$. The radius of gyration is a measure of how 'stretched' or 'curled up' polymers are in the solution. R_g is expected to scale upon increasing polymer length N with the *Flory exponent* ν calculated from renormalization group theory:[2]

$$R_g \propto N^\nu, \nu = \frac{3}{5} \tag{11}$$

2.5 Correctness Checks

The Rosenbluth algorithm along with the PERM modification as presented in section 2.2 and 2.3 performs polymer growth resembling a SAW (Self Avoiding random Walk). The self avoidance follows from the weight of each trial position j: $w_j = e^{-\beta H}$ where $\beta = \frac{1}{k_b T}$. The Boltzmann factor is obtained by calculation of the energy of each trial position due interaction with all existing beads through a Lennard-Jones potential:

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

The Lennard-Jones potential exhibits a minimum at the distance σ. Beads that are more than σ away from the trial position j show attractive interaction and beads that are closer show repulsive interaction. Trial positions with attractive interaction posses negative energies, hence yield large weights. On the contrary, trial positions placed close to existing beads posses large positive energies leading to vanishing weights. Such sampling of polymer growth leads to avoidance of bead positions that are placed too close to already existing bead positions. The polymer avoids itself upon growth. In this case one finds the end-to-end distance to scale as given in equation 9 with $\nu = 0.75$.[3]

When removing interaction, polymer sampling happens entirely random since all trial positions are weighted equally and can be chosen with equal probability This case is called 'random walk' (RW) and is subject to the diffusion limit where a growth of the end-to-end distance is expected to follow equation 9 with $\nu = 0.5$.

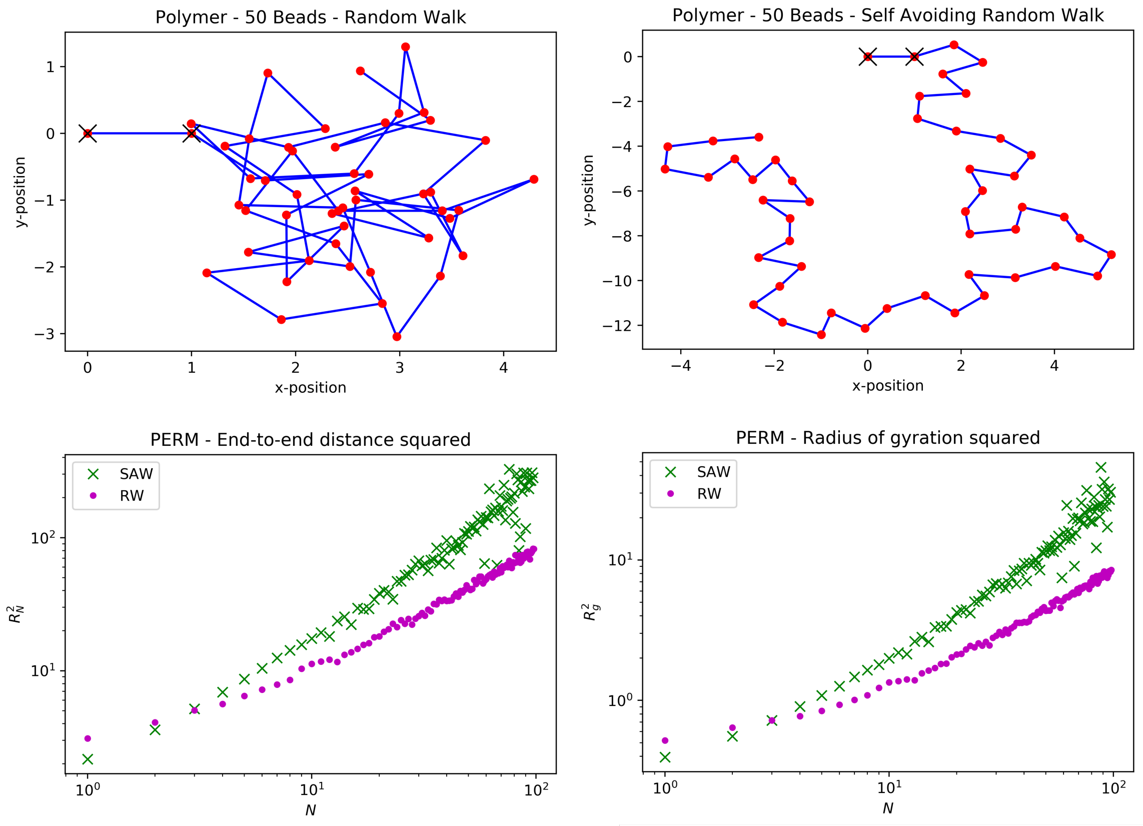

Figure 1: Comparison of random walk (RW) and self avoiding random walk (SAW). Top panels display a single polymer grown up to a length of 50 beads subject to RW (left) and SAW (right), clearly identified by the spacial occupation of each polymer. Bottom panels show the growth of end-to-end distance squared (left) as well as radius of gyration squared (right) vs. the bead length. Data is displayed without errors here.

Figure 1 illustrates the difference in RW and SAW. The two graphs displayed on top show two polymers grown to a length of 50 beads. The left image shows a RW where the polymer crosses itself as the beads do not interact. Whereas a SAW results in a rather stretched out polymer chain depicted on the right. The beads with black crosses are the initial beads from which the sampling begins. A direct comparison of end-to-end distance and radius of gyration vs. bead length is given in the figures below. Both observables grow quicker with bead length for the interacting case. This is already implied by the top images as the spatial occupation for the SAW is greater than for the RW. In fact, a power law fit to the displayed end-to-end distance data leads to $\nu \approx 0.5$ for the RW and $\nu \approx 0.75$ for the SAW as was expected. This verifies the implementation of the algorithm.

3 Results

Polymers were grown up to a length of 250 beads using the Rosenbluth and PERM algorithm. Every new bead was chosen from six trial bead positions placed with equal distance around a circle with a random offset value. The radius was set to one and the origin was set to the current last bead position. The parameters $\sigma = 0.8$ and $\epsilon = T = 1$ to define the Lennard-Jones interaction potential were used uniformly for all subsequent simulations. Those parameters do not define actual energies but rather dictate the interaction of beads in the polymers during the growth process. For the Rosenbluth algorithm 10000 polymers were grown of each polymer length N such that the weighted average of an observable is extracted from a population of 10000 individuals. The PERM algorithm was set up to grow from a population of 500 polymers for each value of N in a similar fashion. The final number of polymers in the population is specified as *population size*. In the subsequent figures only every third bead is displayed as a data point to increase readability of the presented plots. Calculations however, have been performed on the full data sets.

3.1 End-to-end distance

The end-to-end distance was calculated as described in section 2.4. Figure 2 depicts R_N^2 calculated from polymers grown by the Rosenbluth and PERM algorithm. One can clearly see how the results obtained by the Rosenbluth algorithm tend to spread out from a length of 40 to 50 beads. This effect increases with increasing polymer length for which the error of the weighted mean is growing as well. The error was calculated by equation 6. It is for most of the bead numbers a relatively small value which deviates from previous results.[3] Only a few beads show a potentially meaning full error for large N. The suppression of the actual error size is enhanced by presenting the squared data on a logarithmic scale. However, in the here performed simulation, 10000 polymers were grown for each polymer length resulting a weighted average of many contributors which still leads to strong deviations for large N. The spreading from the expected growth rate is due to insufficient cancellation of high-energy configurations. Those structures will contribute to the average but with a low weight. The probability for not well performed SAW becomes larger with increasing polymer length leading to the observed spreading. This is due to the fact that the number of low-weight polymers in the distribution grows exponentially with increasing length. Growing the population into a good sample of this distribution by random sampling therefore becomes extremely exponentially unlikely, making sampling biases the leading source of error for this results.

This limitation can be overcome with the PERM algorithm as shown by the green crosses in the same plot. The dashed red line presents the expected growth rate fitted to the PERM data with a fixed exponent of $\nu = 0.75$. The good agreement also for long polymers underlines the significance of the PERM algorithm. Each population was initialized with 500 polymers which increased to a relatively constant value of 800 to 1000 polymers upon growth.

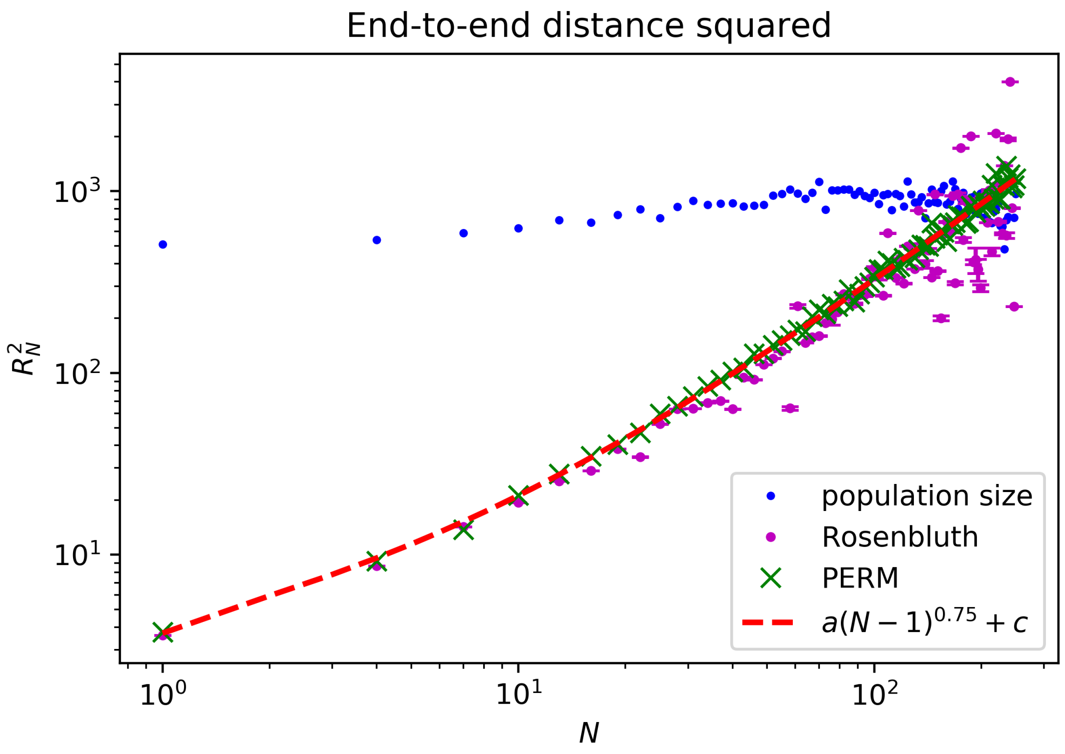

Figure 2: End-to-end distance squared vs. bead length for SAW simulations by Rosenbluth and PERM algorithm. The dashed line is fitted to the data obtained by PERM algorithm and blue dots depict the population size obtained by PERM.

3.2 Radius of gyration

The second observable, radius of gyration squared R_g^2, was obtained as described in section 2.4 and displayed in figure 3. Both, Rosenbluth and PERM data, behave similarly. Again, from 40-50 beads first discrepancies between the two algorithms occur. The red dashed line is fitted to the PERM data in form of the power law described by Teraoka[2]. The obtained exponent reads $b \approx 0.64$ which is close to the expected value of 0.6. Thus far, the PERM simulation resembles the expected growth rate well. The spreading of data points obtained by the Rosenbluth simulation has the same origin as described in the previous section 3.1, since both quantities are dependent on spatial occupation of the polymers. The deviation from the fitted curve visible for small N, is due to the divergence of the fit function towards zero. The radius of gyration for polymers however can never go to zero for a finite number of beads. The expected behaviour is to be regarded as asymptotic behavior, hence it becomes of interest only for larger numbers of beads numbers where the accordance of simulation and theory is indeed satisfied.

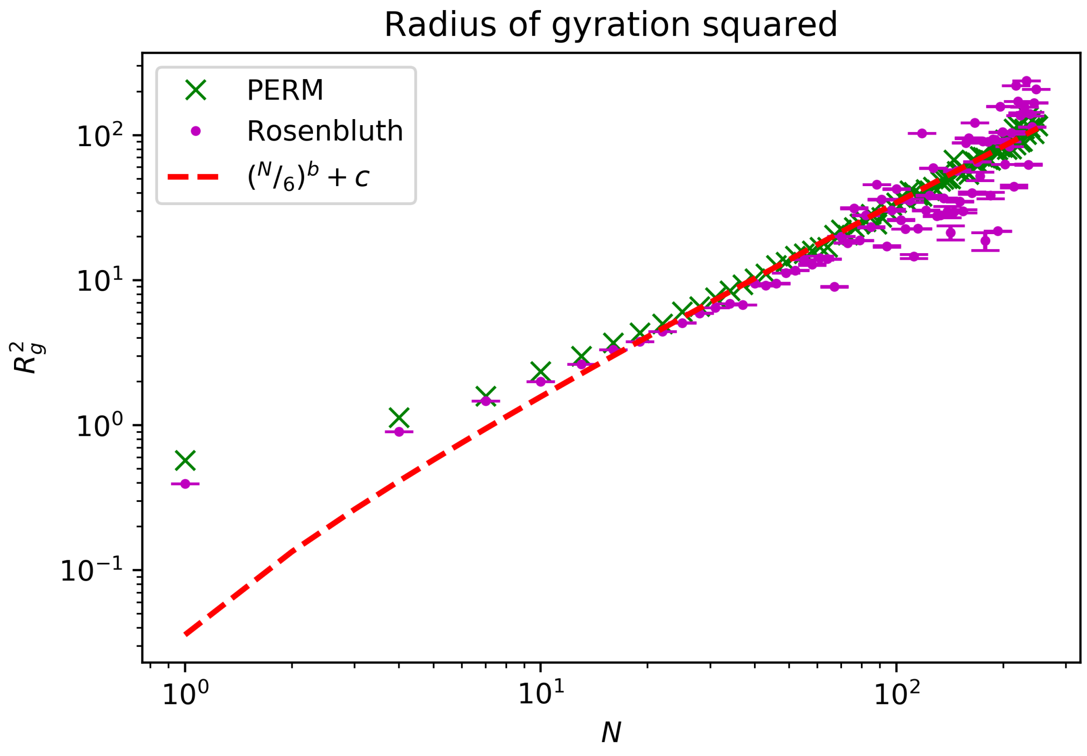

Figure 3: Radius of gyration squared vs. bead length for SAW simulated by Rosenbluth and PERM algorithm.

4 Conclusion

A Python 3 simulation of two-dimensional off-lattice polymers was programmed to compare two variants of Rosenbluth algorithm, part of the Monte Carlo integration methods. The advantage of PERM over the traditional Rosenbluth algorithm was emphasized especially for polymer chain exceeding the length of 40-50 beads. The calculated observables end-to-end distance and radius of gyration are in good agreement with literature. Upon increasing bead length we found the growth exponent $\nu \approx 0.75$ and $\nu \approx 0.64$ for the two observables. This results along with the performed correctness check motivated us to believe that a successful implementation of this project is achieved.

A Source code

All the function used in the simulation are contained in a separate file called `rosenbluth.py`, while the main program, `main.py`, executes the actual simulations. The user can specify the values of interaction parameters T, σ, ϵ, res where the last variable defines the number of trial positions. The parameters `start, bb, step` define the values of the bead length for which simulations are performed. The Rosenbluth and PERM algorithm are run after each other. Both produce data in the same structure as lists of lists in which polymer beads positions with corresponding weights are stored. At last, plots as presented in the report, can be generated obtaining the discussed exponents ν. This way we have a clear and convenient way of changing the simulation setup without the need to access the core of the algorithms directly.

A.1 Performance

Generally, we observed that the PERM algorithm yields more robust results as compared to the Rosenbluth algorithm. Due to the vectorization we were able to simulate vast amounts of polymers using the Rosenbluth algorithm. The here presented data is an average of 2.5 million polymers which took a commercial laptop approximately eight hours working on a single core. Due to the constant change in population size in PERM, we opted for a more flexible data type than numpy arrays that allowed to remove and add population elements effortlessly. With the use of lists we had to sacrifice the aid of complete vectorization although such an approach could greatly benefit from parallelization (not implemented). Nevertheless PERM could provide consistent and correct results with smaller population size, resulting in a faster computation time.

A.2 Work division

Ludwig implemented the first version of the Rosenbluth algorithm which was completely revised under vectorization of the methods by Isacco. From here Ludwig implemented procedures to obtain observables with decent visualization and correctness checks. Meanwhile, Isacco prepared the implementation of the PERM algorithm. A recursive approach was attempted and proved to be working correctly for the Rosenbluth algorithm but its extension to PERM resulted in failure due to the great difficulty in controlling the population size (as explained in Section 2). The current breadth-first approach was then implemented and used to reproduce all the results. Ludwig debugged existing code, planned, executed and visualized the presented results of larger simulations. Almost every step was discussed as a team which is how we wrote the weekly journal entry and the report.

References

[1] Peter Grassberger. “Pruned-enriched Rosenbluth method: Simulations of θ polymers of chain length up to 1 000 000”. In: *Phys. Rev. E* 56 (3 1997), pp. 3682–3693. DOI: 10.1103/PhysRevE.56.3682. URL: https://link.aps.org/doi/10.1103/PhysRevE.56.3682.

[2] Iwao Teraoka. *Polymer Solutions: An Introduction to physical Properties.* John Wiley & Sons, 2002. ISBN: 0-471-22451-0.

[3] J. M. Thijssen. *Computational Physics.* Cambridge University Press, 1998. ISBN: 9780521833462.