Polymer ensemble generation with PERM

S. Bechan(4146425), K. Elgin(4163389), Z. Ramlakhan(4170229)

April 12, 2015

Abstract

In this report we simulated a population of polymers which performed a self-avoiding walk by means of the pruned-enriched Rosenbluth algorithm. However, we didn't employ the standard PERM algorithm as described in the theory, but avoided self-crossings by ending the chains prematurely when the weights in the algorithm equalled zero. This yielded an end-to-end distance (squared) which was overestimated for small chains. This result could be improved by taking the weighted average over the ensemble instead of the regular average.

1 Introduction

A mesoscopic polymer model is one that describes only the major features of the polymer ensemble and where details are of minor importance. In the model, an individual polymer is represented by a chain of beads where the beads represent the monomers of the actual polymer. The beads have a fixed separation distance which is the only interaction between neighbouring beads and remote beads influence each other by means of the van der Waals attraction.
The aim of this study is the generation of an ensemble of polymer configurations which are distributed along the canonical ensemble at any given temperature. Algorithms used to build such a model are based on the Metropolis Monte Carlo method, which is explained in Section 2. Examples are the pivot algorithm or the reptation algorithm.
This study employs the Rosenbluth method and its extension, the pruned-enriched Rosenbluth method. Section 3 explain the Rosenbluth method.
After generating the ensemble the end-to-end distance R of the polymers is evaluated to see whether it adheres to the known scaling with respect to number of beads of the polymer. The results of this quantity is presented in Section 4.

2 The Metropolis method

Configurations in the canonical ensemble should be weighted according to the Boltzmann factor, i.e.

$$\rho(X) \propto exp\left(-\beta E(X)\right) \tag{1}$$

with $\beta = \frac{1}{k_B T}$. The number of configurations with a (high) specific energy increases exponentially with temperature however. This means the algorithm produces high energy configurations that are then rejected anyways. This, of course, is inefficient.
Instead of generating statistically independent configurations, the *Metropolis method[1]*. makes use of *Markov chains*. Before the concept of Markov chains is elaborated, first consider a uncorrelated chain of N objects, $X_1, \ldots, X_N$. The probability of occurrence of a sequence is given by

$$P_N(X_1, X_2, \ldots, X_N) = P_1(X_1)P_2(X_2)\ldots P_N(X_N) \tag{2}$$

where $P_1(X_1)$ is the independent probability of occurrence for object X. A Markov chain is defined in terms of the *transition probability* $T(X \to X')$. The sequence's probability then becomes

$$P_N(X_1, X_2, \ldots, X_N) = P_1(X_1)T(X_1 \to X_2)T(X_2 \to X_3)\ldots T(X_{N-1} \to X_N) \tag{3}$$

The transition probabilities are normalised, i.e.

$$\sum_{X'} T(X \to X') = 1 \tag{4}$$

An example of a Markovian sequence is the random walk where the next step in the walk is independent of the walker's history and thus only depends on his present position.
The algorithm should produce a Markov chain of configurations with a distribution proportional to the Boltzmann factor and this distribution should be independent of the position within the chain and the initial configuration. An *ergodic* Markov chain yields an invariant distribution, for long times. A Markov chain is ergodic if it has the following properties:

- *Connectedness* or *irreducibility*
  Every configuration should be accessible from every other configuration within a finite number of steps.

- The absence of *periodicity*
  It is not possible to return to a particular, previous configuration within a certain number of steps.

The Metropolis method generates a Markov chain of sequences with the desired invariant distribution, in this case the Boltzmann distribution $exp(-\beta E)$.
It is necessary to find $T(X \to X')$ which leads to a stationary distribution $\rho(X)$. $\rho(X, t)$ gives the probability of occurrence X at step t and is ergodic i.e. it is independent of t for large t.
A change in ρ from one step to another is the result of the following two reasons:

- at time t, $X \to X'$ leads to a decrease in ρ.

- at time $t + 1$, $X' \to X$ leads to an increase in ρ.

This phenomenon is summarised in the *master equation*:

$$\rho(X, t+1) - \rho(X, t) = \sum_{X'} T(X' \to X)\rho(X', t) - \sum_{X'} T(X \to X')\rho(X, t) \quad (5)$$

If ρ is stationary, then $\rho(X, t+1) = \rho(X, t)$ must hold. This means that the following is true:

$$\sum_{X'} T(X \to X')\rho(X) = \sum_{X'} T(X' \to X)\rho(X') \quad (6)$$

The t-dependence is omitted as the distribution is now stationary.
The *detailed balance* is a particular solution to this equation:

$$T(X \to X')\rho(X) = T(X' \to X)\rho(X'), \forall X, X' \quad (7)$$

This equation basically says that a decrease in ρ is directly compensated by an increase in ρ equal in magnitude, resulting in no change in ρ whatsoever. Thus, ρ is a stationary distribution.
The transition probabilities are rewritten as follows:

$$\begin{aligned} T(X \to X') &= \omega_{XX'} A_{XX'} \\ \omega_{XX'} &= \omega_{X'X} \\ \sum \omega_{XX'} &= 1 \\ 0 \leq &\omega_{XX'} \leq 1 \\ 0 < &A_{XX'} < 1 \end{aligned}$$

Substituting this into the detailed balances yields

$$\frac{A_{XX'}}{A_{X'X}} = \frac{\rho(X')}{\rho(X)} \quad (8)$$

In the algorithm, $\omega_{XX'}$ is the *trial step probability* and $A_{XX'}$ the *acceptance probability*. The algorithm consists of two stages. Initially, the system is in state X and a new state X' is proposed. X' has probability $\omega_{XX'}$. In the second stage the old and new weights, $\rho(X)$ and $\rho(X')$ respectively, are compared. If $\rho(X') > \rho(X)$, $A_{XX'} = 1$ and the new state X' is accepted. If $\rho(X') < \rho(X)$, X' is accepted with probability $A_{XX'} = \frac{\rho(X')}{\rho(X)}$. Note that $A_{XX'}$ satisfies (8), X' is accepted with $A_{XX'}$ and rejected with $1 - A_{XX'}$.

The algorithm is executed by generating a random number r within the interval $[0, 1]$. Then

$$\begin{aligned} r < A_{XX'} \qquad & X \to X' \\ r > A_{XX'} \qquad & X \to X \end{aligned}$$

3 The Rosenbluth method

The Rosenbluth method generates polymers by continually adding beads to an initial two-bead polymer until the maximum number of beads has been reached. New segments are added to the polymer in a way that avoids high energy conformations and crossings with this method.
With this method, polymers have the following characteristics:

- A polymer has a total of N beads, represented as point particles. The separation between two neighbouring beads is fixed. This fixed distance is the only interactions neighbouring beads experience.
- Remote beads experience a Lennard-Jones interaction. This is given by

$$U = 4\epsilon \left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6} \right] \tag{9}$$

  where ϵ is the depth of the potential well, σ the finite distance at which the inter-particle potential is zero and r is the distance between two particles.

The Rosenbluth method starts by initialising the aforementioned initial two-bead polymer which is simply a polymer with one bead at the origin and another at $(1, 0)$. Note that the bead separation distance is taken to be 1. According to the Metropolis method, the third bead should obey the distribution $exp\left(\frac{-E(\theta)}{k_B T}\right)$ where $E(\theta)$ is the interaction energy of the third bead with the first two. To minimize the number of trial steps the θ-space is discretised to obtain j values for θ, with spacing $\frac{2\pi}{j}$ and a random offset.
The sum of the weights are given by

$$W^{(l)} = \sum_j w_j^{(l)} = \sum_j exp\left(\frac{-E(\theta_j)}{k_B T}\right) \tag{10}$$

where l denotes the bead that is added. θ_j is accepted with probability $\frac{w_j^{(l)}}{W^{(l)}}$. The process of choosing θ_j begins by dividing the interval of $[0, 1]$ into j segments of size $\frac{w_j^{(l)}}{W^{(l)}}$. A random number in the same interval of $[0, 1]$ is generated and the algorithm checks which segment j it belongs to. This results in the angle θ_j the third bead should make with the other two. The way θ_j is selected is called the *roulette wheel selection method*.
The final Boltzmann weight of the polymer is given by

$$exp\left(\frac{-E_{total}}{k_B T}\right) = \prod_{l=3}^{N} w_j^{(l)} \tag{11}$$

where j denotes the choice made for θ at step l.
The actual probability for finding the system in a particular configuration is

$$P \simeq \prod_{l=3}^{N} \frac{w_j^{(l)}}{W_j^{(l)}} \tag{12}$$

Thus, compared to (11), P is off by a factor W_j^l. This is compensated by storing this number into a weight factor for calculating properties of the polymers.
The second method for compensating for this factor is by assigning each θ the same probability equal to $\frac{1}{N_\theta}$. The weights of each polymer is then equal to $\prod_{l=1}^{N} w_j^{(l)}$.

The Rosenbluth method can be summarised with the following pseudocode:[1]

```
Set PolWeight to 1;
ROUTINE Addbead(Polymer,PolWeight,L)
        Calculate the weights w_j^L and their product W^L;
        Add bead number L;
        PolWeight = PolWeight*W^L;
        IF L<N THEN
                AddBead(Polymer,PolWeight,L+1);
        END IF
END ROUTINE AddBead
```

An interesting quantity to consider is the end-to-end distance R of a polymer which is the distance between the two end points. R is related to N as follows:

$$R \propto N^\nu \tag{13}$$

with $\nu = 0.75$ in 2D. Figure 1 shows R^2 as a function of N on a log-log scale with cross markers. Note that the Rosenbluth method faithfully produces polymers obeying this relation for polymers up to a 100 beads or so. After that fluctuations of R^2 are evident. This is due to the fact that the algorithm does not sufficiently suppress high energy configurations which leads to those fluctuations.

The *pruned-enriched Rosenbluth method* (PERM) remedies this issue by removing "bad" configurations from the populations and replacing them with copies of the "good" ones which independently evolve further. PERM prunes the population by removing polymers with low weight (weak polymers) and enriches it by multiplying the polymers with large weight (strong polymers).
Figure 1 shows the results of the PERM algorithm with double crosses and shows that even after 100 beads, the result agrees with the fit.

Consider a polymer with a relatively large weight. This polymer needs to be enriched. To do so, two copies are made of this polymer. The copies are called 'children' and have half the weight of their 'parent' to keep the total weight constant. An upper limit UpLim is defined such that polymers with

[1] This and any following pseudocode has directly been taken from [2] without modification.

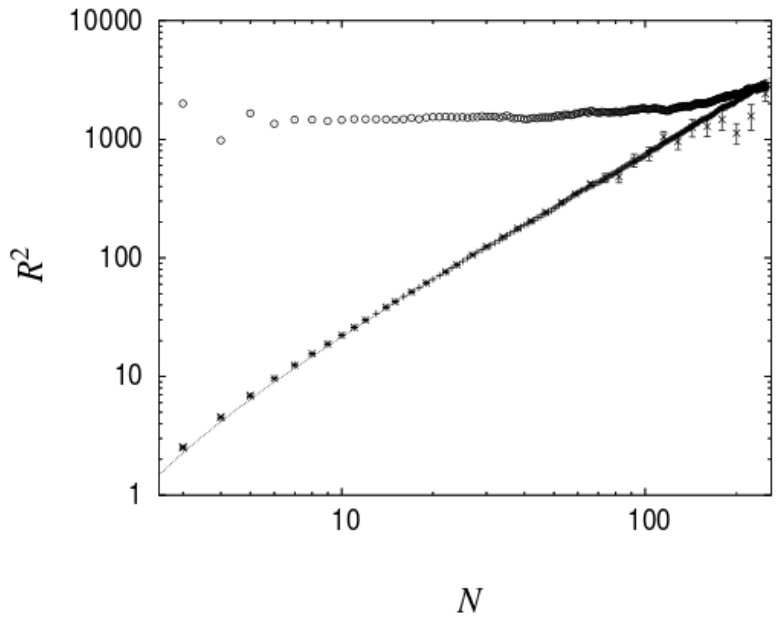

Figure 1: R^2 as function of N on a log-log scale. The cross markers are the result of the Rosenbluth algorithm, the doubble cross markers are the result of the PERM algorithm. The circles represent population sizes in the PERM algorithm. The fit has the form $a(N-1)^{1.5}$ [2].

weight exceeding UpLim are enriched. The Rosenbluth pseudocode would then be extended by the following lines:

```
IF (PolWeight > UpLim) THEN
        NewWeight=0.5*PolWeight;
        AddBead(Polymer, NewWeight,L+1);
        NewWeight=0.5*PolWeight;
        AddBead(Polymer, NewWeight,L+1);
END IF
```

Pruning removes polymers from the population, but this has to be executed in a way that leaves the distribution unchanged. Therefore weak polymers are removed with probability 0.5 and multiplying the weight of the remaining polymers with a factor 2. Here a limit LowLim is defined such that polymers with weight below LowLim are pruned. The Rosenbluth pseudocode is extended with the following lines:

```
IF (PolWeight < LowLim) THEN
        Choose random number R uniformly between 0 and 1 ;
        IF (R < 0.5) THEN
                NewWeight = 2*PolWeight;
                AddBead(Polymer, NewWeight,L+1)
        END IF;
END IF
```

UpLim and LowLim determine whether the population will grow, shrink or remain the same. The limits depend on the average weight AvWeight at step L. The average is updated when a few polymers reach this length. the limits are takes as multiples of the ratio of the average and the weight Weight3 of the polymer with the shortest length. This would be the polymer at the first step with three beads.
The enrichment limit is expressed as:

$$\text{UpLim} = \alpha * \frac{\text{AvWeight}}{\text{Weight3}} \tag{14}$$

The same holds for LowLim. Good values for α in UpLim and LowLim are 2 and 1.2, respectively. α may depend on the level L. This dependence is removed by multiplying the polymer weight by a factor $\frac{1}{0.75N_\theta}$ with N_θ the number of values for θ.

3.1 Extension to 3D

The implementation of the Rosenbluth algorithm that is used in this code is two dimensional. Most obviously of all, the length-by-2 array for the bead position would have to be converted to a length-by-3 array with the z-position as the third column. The two starting beads would be positioned at (0,0,0) and (1,0,0). Moreover, calculations such as the end to end distance and the Lennard-Jones potential would need to be rewritten to work for three dimensions. The interesting part of a conversion to 3D is deciding how the position of the new bead is selected, since we can no longer just use a number of equally spaced points on a circle of 2π. The simplest way of expanding this bead position code to three dimensions would be to add a second selector interval for the z-axis. On top of the equally spaced points on a circle of 2π that we use to calculate x- and y-positions, we would use a selector interval of $[0,\pi]$ and once again take a number of points equally spaced from each other on this interval. This interval would then be used to calculate the z-position of the bead. The change in z-position of the new bead compared to the previous bead would then be the sine of the angle on the $[0,\pi]$ interval. Adding this z-position change to the x- and y-positions changes already calculated in the code would give us a 3D space for the bead to move in. This would greatly increase the number of bead position for every step. Assuming we have n1 different angles on the 2π circle and n2 different angles on the $[0,\pi]$ interval, the new bead will have n1*n2

possible positions. A problem of selecting beads in this manner is that we no longer have a truly equally spaced distribution. A 3D sphere with a radius of 1 would have all possible 3D bead positions on it. Obviously the beads near the center of the sphere will be further apart from each other than beads at the top of the sphere. In order to counter this problem, a different more complicated algorithm would have to be used. This algorithm would have to be able to calculate how to equally space n1*n2 points over a sphere, given the starting location of a single bead on this sphere.

4 Results

4.1 Pruning in the PERM algorithm

The method of pruning described in the theory is not the method used in this specific algorithm. The reason for this is that when using the variable alphadown pruning as described for normal PERM, it was found that the entire population of polymers was over pruned meaning next to no actual finished polymers survived. This was a phenomenon that we could not combat by changing either the alphadown value or the way we calculated the weights. The alternate way of pruning used in this code is a simpler variant that catches less weak polymers but still removes out the weakest strains. If a polymer has a weight of 0 (which means it is not actually 0 but so small that python calculates it as 0), that polymer chain is prematurely ended. A weight of 0 is equivalent to a position where no matter where the new bead is placed there will be some form of overlap or close proximity to another part of the chain, leading to a high Lennard-Jones potential and therefore a weight of 0. By doing this the weakest, least viable chains are removed from the list. While obviously not as viable as the variable lower limit method of standard PERM, this method of pruning still removes some weak strains from the collection meaning it still has use in the algorithm as a way to prune in the simulation.

4.2 Population size versus polymer size

When looking at the polymer size in the figure below we see that generally from our starting size of polpop polymers, we see an expansion of the number of polymers, as strong chains are multiplied leading to an increase in the total number of polymers in the population. The longer the chains become however, the greater the chance becomes that one of the chains reaches a limit where no matter what the next bead position is there is a zero chance of the bead existing leading to a total weight of 0 and therefore pruning. The effect of this pruning starts to compensate for the effect of multiplication and given long enough chains, the population size seems to stabilize and sometimes even decrease as the longer and longer chains lead to higher and higher chances of W=0 endpoints and therefore pruning.

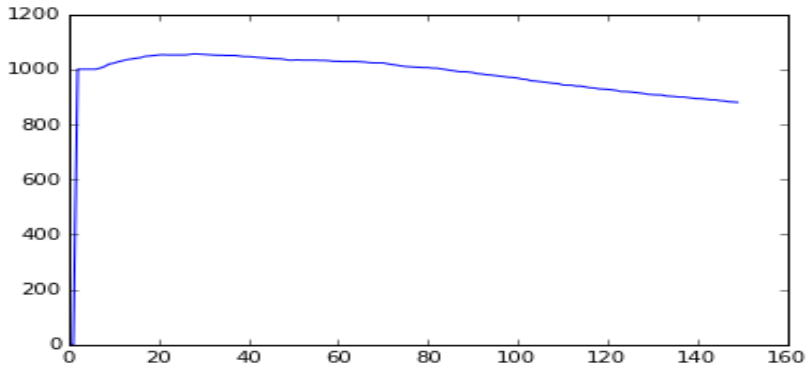

Figure 2: Polymer size (x-axis) plotted against the polymer population (y-axis). Notice the saturation and the eventual decrease for longer chains

In this report 100 different polymer chains were generated through the Rosenbluth algorithm and prune - enrichment method. Some of these polymers which were created, can be seen in the figure below.

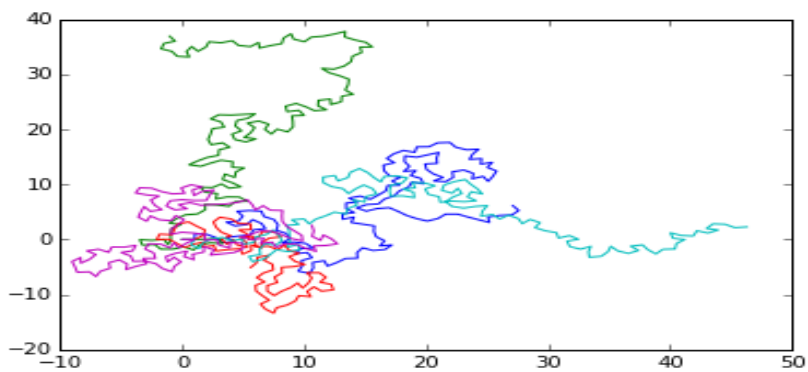

Figure 3: Five typical polymer generated consisting of 150 beads. Along the horizontal axis is displayed the x-coordinate, along the vertical axis the y-coordinate. Notice the lack of self-crossings.

When looking at the figure above picturing 5 polymers, what's noticeable are the lack of self-crossings. The self-crossings are possible since we assumed that each bead to be added only interacts with it's nearest neighbour. However, these are high-energy configurations for which the weights are near zero. This means that they are less probable to occur. When a self-crossing would occur the chain was prematurely ended. In this way the polymers follow a self-avoiding random walk.

When looking at the end-to-end distance (squared) which is plotted below versus the number of beads, it is clear that the obtained data deviates from the theoretical fit especially for shorter chains. This might be due to the fact that we averaged over the entire population of polymers instead of taking the weighted average.

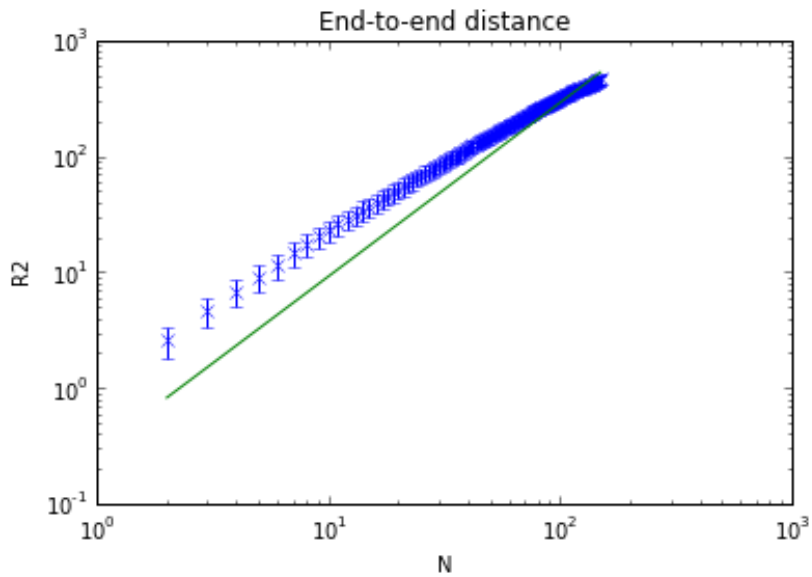

Figure 4: End-to-end distance squared versus number of beads and a polynomial fit of the form $aN^{1.5}$. Along the horizontal axis is displayed the number of beads, N, along the vertical axis the end-to-end distance.

5 Conclusion

In this report an ensemble of 100 polymers was simulated by using the Rosenbluth algorithm complemented with the pruned-enriched method. Important assumptions made were the fact that the only interaction was between neighbouring beads (the Lennard-Jones interaction) and that neighbouring beads have a fixed separation distance. The chains which included self-crossings (e.g. with weight zero) were prematurely ended. In this way the population of polymers follows a self-avoiding random walk. It was found that the square of the end-to-end distance as a function of the length of the chains (the number of beads) was overestimated in comparison with the theoretical result especially for shorter chains. This might be improved by taking the weighted average over the polymer population instead of the normal average. Also the fact that

the chains were prematurely ended if the weight would be zero, could result in higher end-to-end distances for a given chain length.

References

[1] Nicholas Metropolis et al. "Equation of state calculations by fast computing machines". In: *The journal of chemical physics* 21.6 (1953), pp. 1087–1092.

[2] Jos Thijssen. *Computational physics*. Cambridge University Press, 2007.

6 Appendix

Link to our code:
https://github.com/KenElg/Polymers/tree/master
Instructions: run main.py