Statistische Mechanik Bonusblatt

Jun Wei Tan*

Julius-Maximilians-Universität Würzburg

(Dated: December 20, 2024)

CONTENTS

I. AUFGABE 1

Betrachten Sie eine Kette aus N Monomeren der Größe a in zwei Dimensionen. D.h., jedem Glied i kann der Vektor $\vec{r}_i$ zugeordnet werden, der in eine beliebige Richtung zeigen kann und die Länge $|\vec{r}_i| = a$ hat.

a) Berechnen Sie für dieses System die mittlere quadratische Distanz zwischen Anfangs- und Endpunkt:

$$\langle \vec{R}^2 \rangle = \left\langle \left(\sum_{i=1}^{N} \vec{r}_i \right)^2 \right\rangle$$

Hinweis: Hier lässt sich mit Symmetrie argumentieren. Was ist $\langle \vec{r}_i \cdot \vec{r}_j \rangle$ für $i = j$ bzw. für $i \neq j$?

* jun-wei.tan@stud-mail.uni-wuerzburg.de

b) Das Polymer befindet sich jetzt in einem Wärmebad mit der Temperatur T. Die beiden Enden des Polymers werden mit einer Kraft $\vec{F} = (0, 0, F)$ auseinandergezogen, so dass (1) gilt. Berechnen Sie die Korrelationsfunktionen

$$\langle \vec{R} \rangle = \left\langle \sum_{i=1}^{N} \vec{r}_i \right\rangle ,$$

$$\langle \vec{R}^2 \rangle = \left\langle \left(\sum_{i=1}^{N} \vec{r}_i \right)^2 \right\rangle$$

als Funktion der Kraft und der Temperatur. Wie verhalten sich diese in den Grenzfällen $aF \ll k_B T$ und $aF \gg k_B T$?

Proof. a) Für $i \neq j$ sind $\vec{r}_i$ und $\vec{r}_j$ unabhängig. Damit ist

$$\langle \vec{r}_i \cdot \vec{r}_j \rangle = \langle \vec{r}_i \rangle \cdot \langle \vec{r}_j \rangle = \vec{0} \cdot \vec{0} = 0.$$

Für $i = j$ ist $\vec{r}_i \cdot \vec{r}_j = |\vec{r}_i|^2 = a^2$, was überhaupt nicht stochastisch ist und damit ist $\langle \vec{r}_i \cdot \vec{r}_i \rangle = a^2$. Damit ist

$$\begin{aligned}
\langle \vec{R}^2 \rangle &= \left\langle \left(\sum_{i=1}^{N} \vec{r}_i \right)^2 \right\rangle \\
&= \left\langle \sum_{i=1}^{N} \sum_{j=1}^{N} \vec{r}_i \cdot \vec{r}_j \right\rangle \\
&= \sum_{i=1}^{N} \sum_{j=1}^{N} \langle \vec{r}_i \cdot \vec{r}_j \rangle \\
&= \sum_{i=1}^{N} \sum_{j=1}^{N} a^2 \delta_{ij} \\
&= N a^2.
\end{aligned}$$

□

II. AUFGABE 2

Appendix A: Code

1. Code für den Random Walk

```cpp
#include <cstdio>
#include <vector>
#include <random>
#include <math.h>
#include <fstream>
#include <iostream>

inline int sqDist(std::pair<int,int> coord){
  return coord.first*coord.first + coord.second*coord.second;
}

class RandomWalk {
  public:
    int N; //Length
    std::vector<std::pair<int,int>> coords; //list of positions
    std::pair<int,int> currentPos;

    //random device
    static std::random_device rd;
    /* This may not be a random seed!
     * https://en.cppreference.com/w/cpp/numeric/random/random_device
     * On my computer, it generated the same numbers at every run
     * Maybe it works better on yours lmao
     */
    static std::mt19937 gen; //static so different class instances generate different numbers
      static std::discrete_distribution<> dist;

    RandomWalk(int Np){
        this -> N = Np;
```

```
    }

    int generateWalk(){
    //reset variables
    currentPos = std::make_pair(0,0);
    coords = std::vector<std::pair<int,int>>();
    coords.push_back(currentPos);

    for (int i = 0; i < N; ++i){
      int dir = dist(gen); //0 for right, 1 for up, 2 for left, 3 for ↩
        down
      switch (dir){
        case 0:
          currentPos.first++;
          break;
        case 1:
          currentPos.second++;
          break;
        case 2:
          currentPos.first--;
          break;
        case 3:
          currentPos.second--;
      }
      coords.push_back(currentPos);
    }
    return 0; //no errors; this will be different in the SARW
    }
};

```

```
//Initialize static random members
std::random_device RandomWalk::rd;
std::mt19937 RandomWalk::gen(rd());
std::discrete_distribution<> RandomWalk::dist({1,1,1,1});


int main(){
    int numExperiments = 1000000;
    int L = 200; //length of a single walk
    std::vector<std::vector<int>> distances(numExperiments, std::vector<int>(L+1,0));
    RandomWalk rw = RandomWalk(L);
    //we export the first 20 trajectories here, 20 is hardcoded
    std::ofstream outfile;
    outfile.open("trajectories.csv");
    for (int exp = 0; exp < numExperiments; ++exp){
        rw.generateWalk();
        for (int i = 1; i <= L; ++i) distances[exp][i] = sqDist(rw.coords[i]);
        if (exp < 20) for (int i = 1; i <= L; ++i) outfile << rw.coords[i].first << "," << rw.coords[i].second << "\n";
    }
    outfile.close();

    /*for (int exp = 0; exp < numExperiments; ++exp){
for (int i = 0; i <= L; ++i) printf("%d ", distances[exp][i]); printf("\n");
    }*/

    /* Be careful when taking the average/stddev for large enough numbers
     * Because we compute the sum before dividing, we may have an integer overflow
```

```
     * To make this as high as possible, we use a long double
     */

    std::vector<double> averageDistances(L+1,0);
    std::vector<double> standardDeviations(L+1,0);
    for (int i = 1; i <= L; ++i){
        long double sum = 0.0;
        for (int exp = 0; exp < numExperiments; ++exp) sum += distances[exp][i];
        averageDistances[i] = sum / numExperiments;
        sum = 0.0;
        for (int exp = 0; exp < numExperiments; ++exp) sum += pow(distances[exp][i] - averageDistances[i], 2);
        sum /= numExperiments - 1;
        sum = sqrt(sum);
        standardDeviations[i] = sum;
    }

    outfile.open("randomWalk.csv");
    for (int i = 0; i <= L; ++i) outfile << averageDistances[i] << ",";
    outfile << "\n";
    for (int i = 0; i <= L; ++i) outfile << standardDeviations[i] << ",";
    outfile.close();
        return 0;
}
```

2. Code für den Self Avoiding Random Walk

```
#include <cstdio>
```

```
#include <vector>
#include <random>
#include <math.h>
#include <fstream>
#include <iostream>
#include <set>

inline int sqDist(std::pair<int,int> coord){
    return coord.first*coord.first + coord.second*coord.second;
}

class RandomWalk {
    public:
        int N; //Length
        std::vector<std::pair<int,int>> coords; //list of positions
        std::pair<int,int> currentPos;
        std::set<std::pair<int,int>> visited;

        //random device
        static std::random_device rd;
        /* This may not be a random seed!
         * https://en.cppreference.com/w/cpp/numeric/random/random_device
         * On my computer, it generated the same numbers at every run
         * Maybe it works better on yours lmao
         */
        static std::mt19937 gen; //static so different class instances ↩
            generate different numbers
          static std::discrete_distribution<> dist;

        RandomWalk(int Np){
```

```
        this -> N = Np;
    }

    int generateWalk(){
    //reset variables
    currentPos = std::make_pair(0,0);
    coords = std::vector<std::pair<int,int>>();
    visited = std::set<std::pair<int,int>>();

    coords.push_back(currentPos);
    visited.insert(currentPos);

    for (int i = 0; i < N; ++i){
      int dir = dist(gen); //0 for right, 1 for up, 2 for left, 3 for down
      switch (dir){
        case 0:
          currentPos.first++;
          break;
        case 1:
          currentPos.second++;
          break;
        case 2:
          currentPos.first--;
          break;
        case 3:
          currentPos.second--;
      }
      coords.push_back(currentPos);
      if (visited.count(currentPos)) return 1;
```

```
        }
        return 0;
        }

        int generateAvoidingWalk(){
        while (true){
            int x = generateWalk();
            if (not x) return 0;
        }
        }
};

//Initialize static random members
std::random_device RandomWalk::rd;
std::mt19937 RandomWalk::gen(rd());
std::discrete_distribution<> RandomWalk::dist({1,1,1,1});


int main(){
    int numExperiments = 1000000;
    int L = 40; //length of a single walk
    std::vector<std::vector<int>> distances(numExperiments, std::vector<int>(L+1,0));
    RandomWalk rw = RandomWalk(L);
    //here we also export the first 20 walks, 20 is hardcoded
    std::ofstream outfile;
    outfile.open("trajectoriesAvoiding.csv");
    for (int exp = 0; exp < numExperiments; ++exp){
        rw.generateAvoidingWalk();
        for (int i = 1; i <= L; ++i) distances[exp][i] = sqDist(rw.coords[i]);
```

```
    if (exp < 20) for (int i = 1; i <= L; ++i) outfile << rw.coords[i].first << "," << rw.coords[i].second << "\n"; //here is the exporting of the trajectory
  }
  outfile.close();

  /*for (int exp = 0; exp < numExperiments; ++exp){
for (int i = 0; i <= L; ++i) printf("%d ", distances[exp][i]); printf("\n");
  }*/

  /* Be careful when taking the average/stddev for large enough numbers
   * Because we compute the sum before dividing, we may have an integer overflow
   * To make this as high as possible, we use a long double
   */

  std::vector<double> averageDistances(L+1,0);
  std::vector<double> standardDeviations(L+1,0);
  for (int i = 1; i <= L; ++i){
    long double sum = 0.0;
    for (int exp = 0; exp < numExperiments; ++exp) sum += distances[exp][i];
    averageDistances[i] = sum / numExperiments;
    sum = 0.0;
    for (int exp = 0; exp < numExperiments; ++exp) sum += pow(distances[exp][i] - averageDistances[i], 2);
    sum /= numExperiments - 1;
    sum = sqrt(sum);
    standardDeviations[i] = sum;
  }
```

```
    outfile.open("randomAvoidingWalk.csv");
    for (int i = 0; i <= L; ++i) outfile << averageDistances[i] << ",";
    outfile << "\n";
    for (int i = 0; i <= L; ++i) outfile << standardDeviations[i] << ",";
    outfile.close();
        return 0;
}
```

Appendix B: Simulationsdaten

1. Daten für den Random Walk

Zeit (t)	$\langle R(t)^2 \rangle$
0	0 ± 0
1	1 ± 0
2	1.9989 ± 0.0014
3	2.9985 ± 0.0025
4	3.9988 ± 0.0035
5	5.0010 ± 0.0045
6	6.0039 ± 0.0055
7	7.0004 ± 0.0065
8	8.0005 ± 0.0075
9	9.0027 ± 0.0085
10	9.9954 ± 0.0095
11	10.996 ± 0.010
12	11.991 ± 0.011

13	12.995 ± 0.012
14	13.996 ± 0.013
15	14.992 ± 0.014
16	16.003 ± 0.015
17	17.003 ± 0.016
18	18.013 ± 0.018
19	19.008 ± 0.019
20	20.015 ± 0.020
21	21.017 ± 0.021
22	22.022 ± 0.022
23	23.028 ± 0.023
24	24.030 ± 0.024
25	25.032 ± 0.025
26	26.029 ± 0.026
27	27.029 ± 0.027
28	28.030 ± 0.028
29	29.026 ± 0.029
30	30.031 ± 0.030
31	31.035 ± 0.031
32	32.033 ± 0.032
33	33.031 ± 0.033
34	34.036 ± 0.034
35	35.028 ± 0.035

36	36.040 ± 0.036
37	37.022 ± 0.037
38	38.014 ± 0.038
39	39.027 ± 0.039
40	40.032 ± 0.040
41	41.047 ± 0.041
42	42.035 ± 0.042
43	43.022 ± 0.042
44	44.019 ± 0.043
45	45.017 ± 0.044
46	46.013 ± 0.045
47	47.004 ± 0.046
48	48.004 ± 0.047
49	49.010 ± 0.048
50	50.014 ± 0.049
51	51.015 ± 0.050
52	52.027 ± 0.051
53	53.012 ± 0.052
54	54.004 ± 0.053
55	55.010 ± 0.054
56	56.000 ± 0.055
57	57.006 ± 0.056
58	57.997 ± 0.057
59	58.990 ± 0.058

60	60.016 ± 0.059
61	61.009 ± 0.060
62	62.004 ± 0.061
63	63.015 ± 0.062
64	64.016 ± 0.063
65	65.010 ± 0.064
66	66.013 ± 0.065
67	67.037 ± 0.066
68	68.036 ± 0.068
69	69.043 ± 0.069
70	70.039 ± 0.070
71	71.059 ± 0.071
72	72.070 ± 0.072
73	73.074 ± 0.073
74	74.068 ± 0.074
75	75.079 ± 0.075
76	76.103 ± 0.076
77	77.095 ± 0.077
78	78.113 ± 0.078
79	79.104 ± 0.079
80	80.109 ± 0.080
81	81.099 ± 0.081
82	82.111 ± 0.082
83	83.120 ± 0.083

84	84.103 ± 0.084
85	85.121 ± 0.085
86	86.109 ± 0.086
87	87.108 ± 0.087
88	88.118 ± 0.088
89	89.110 ± 0.089
90	90.104 ± 0.090
91	91.095 ± 0.091
92	92.099 ± 0.092
93	93.095 ± 0.093
94	94.104 ± 0.094
95	95.111 ± 0.095
96	96.109 ± 0.096
97	97.110 ± 0.097
98	98.100 ± 0.098
99	99.091 ± 0.099
100	100.091 ± 0.10
101	101.10 ± 0.10
102	102.07 ± 0.10
103	103.06 ± 0.10
104	104.06 ± 0.10
105	105.05 ± 0.10
106	106.04 ± 0.11
107	107.05 ± 0.11

108	108.04 ± 0.11
109	109.05 ± 0.11
110	110.07 ± 0.11
111	111.07 ± 0.11
112	112.07 ± 0.11
113	113.07 ± 0.11
114	114.04 ± 0.11
115	115.04 ± 0.11
116	116.07 ± 0.12
117	117.06 ± 0.12
118	118.03 ± 0.12
119	119.05 ± 0.12
120	120.06 ± 0.12
121	121.06 ± 0.12
122	122.09 ± 0.12
123	123.11 ± 0.12
124	124.11 ± 0.12
125	125.11 ± 0.12
126	126.10 ± 0.13
127	127.10 ± 0.13
128	128.08 ± 0.13
129	129.10 ± 0.13
130	130.11 ± 0.13
131	131.14 ± 0.13

132	132.12 ± 0.13
133	133.11 ± 0.13
134	134.10 ± 0.13
135	135.08 ± 0.13
136	136.08 ± 0.14
137	137.07 ± 0.14
138	138.08 ± 0.14
139	139.08 ± 0.14
140	140.08 ± 0.14
141	141.10 ± 0.14
142	142.10 ± 0.14
143	143.13 ± 0.14
144	144.17 ± 0.14
145	145.19 ± 0.14
146	146.20 ± 0.15
147	147.19 ± 0.15
148	148.18 ± 0.15
149	149.20 ± 0.15
150	150.19 ± 0.15
151	151.19 ± 0.15
152	152.17 ± 0.15
153	153.15 ± 0.15
154	154.14 ± 0.15
155	155.14 ± 0.15

156	156.14 ± 0.16
157	157.12 ± 0.16
158	158.10 ± 0.16
159	159.10 ± 0.16
160	160.07 ± 0.16
161	161.06 ± 0.16
162	162.10 ± 0.16
163	163.10 ± 0.16
164	164.09 ± 0.16
165	165.09 ± 0.16
166	166.09 ± 0.17
167	167.09 ± 0.17
168	168.10 ± 0.17
169	169.09 ± 0.17
170	170.09 ± 0.17
171	171.07 ± 0.17
172	172.06 ± 0.17
173	173.05 ± 0.17
174	174.07 ± 0.17
175	175.08 ± 0.17
176	176.10 ± 0.18
177	177.10 ± 0.18
178	178.13 ± 0.18
179	179.16 ± 0.18

180	180.15 ± 0.18
181	181.15 ± 0.18
182	182.14 ± 0.18
183	183.12 ± 0.18
184	184.13 ± 0.18
185	185.17 ± 0.18
186	186.17 ± 0.19
187	187.15 ± 0.19
188	188.16 ± 0.19
189	189.17 ± 0.19
190	190.20 ± 0.19
191	191.21 ± 0.19
192	192.21 ± 0.19
193	193.22 ± 0.19
194	194.21 ± 0.19
195	195.23 ± 0.19
196	196.25 ± 0.20
197	197.23 ± 0.20
198	198.22 ± 0.20
199	199.19 ± 0.20
200	200.18 ± 0.20

2. Daten für den Self Avoiding Random Walk

Zeit (t)	$\langle R(t)^2 \rangle$

0	0 ± 0
1	1 ± 0
2	2.72338 ± 0.00096
3	4.1670 ± 0.0025
4	5.6479 ± 0.0036
5	7.0419 ± 0.0049
6	8.4476 ± 0.0060
7	9.8035 ± 0.0072
8	11.1628 ± 0.0083
9	12.4847 ± 0.0095
10	13.812 ± 0.011
11	15.107 ± 0.012
12	16.403 ± 0.013
13	17.658 ± 0.014
14	18.925 ± 0.015
15	20.172 ± 0.016
16	21.419 ± 0.017
17	22.641 ± 0.018
18	23.858 ± 0.019
19	25.064 ± 0.021
20	26.262 ± 0.022
21	27.448 ± 0.023
22	28.636 ± 0.024

23	29.791 ± 0.025
24	30.952 ± 0.026
25	32.100 ± 0.027
26	33.236 ± 0.028
27	34.365 ± 0.029
28	35.492 ± 0.030
29	36.594 ± 0.031
30	37.713 ± 0.032
31	38.810 ± 0.033
32	39.898 ± 0.034
33	40.980 ± 0.036
34	42.057 ± 0.037
35	43.113 ± 0.038
36	44.160 ± 0.039
37	45.212 ± 0.040
38	46.239 ± 0.041
39	47.259 ± 0.042
40	48.285 ± 0.043